GUARDIAN AUDITS

SMART CONTRACT SECURITY AUDIT OF

UMAMI

Summary

Audit Firm Guardian

Client Firm Umami Finance

Prepared By Daniel Gelfand, Owen Thurm, Kiki, Kristian Apostolov, ABA, 0xKato

Final Report Date January 10, 2024

Audit Summary

Umami Finance engaged Guardian to review the security of its GMX V2 market index, GMI. From the 11th of December to the 29th of December, 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 9 High/Critical issues were uncovered and promptly remediated by the Umami Finance team. Several issues impacted the fundamental behavior of the protocol, following their remediation Guardian believes the protocol to uphold the functionality described for the GMI product.

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.

Notice that the examined smart contracts are not resistant to internal exploit. 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/UmamiPoCs

Table of Contents

Project Overview

Project Summary

Project Name	Umami Finance
Language	Solidity
Codebase	https://github.com/UmamiDAO/V3-Vaults
Commit(s)	3496063b45c82c92037e07b5a6cb5cbcbd453727

Audit Summary

Delivery Date	January 10, 2024
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	3	0	0	0	0	3
High	6	0	0	0	0	6
Medium	8	0	0	1	0	7
Low	41	0	0	20	0	21

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
AUTH	Auth.sol	653936a83848b8325fce2da9f08ea319851fe067
GFR	GmxFeeReader.sol	0f98e0bfdb11de542d7f284124409221e1698cd8
NPT	NettedPositionTracker.sol	39317f285b21602c5dbb2449f2a48a3053c713a1
PRC	Pricing.sol	73fc709fe089b7940d06d8d46d1b6f4e67b3dc3e
GMXS	GmxStorage.sol	acccaa2fe6465f8da41aeb4572fb0236b07c8f6c
MULC	Multicall.sol	b51b01e897163e732b94936c6374877065944c99
DGC	Delegatecall.sol	b6835202f0301fda4e1881a145ed7f7f812c44b4
LAVU	LibAggregateVaultUtils.sol	3656066ef99cfb2055b827ca5d8258b8a315bf0a
LCY	LibCycle.sol	be53cb15465feb572af892b0e5ed6cbc0659085e
PRCST	PriceCast.sol	f9a86996a7edb3548e45eb0ca52c2b0e8d8b81d1
NETM	NettingMath.sol	47b3232a8396b7684f9c288fae2fe84c5d3e1989
AGVS	AggregateVaultStorage.sol	33adb51555568fd217f11690364006165bec3454
AV	AssetVault.sol	19b4927edd3b5a5fe970c95bb14e57130983fc3e
AGV	AggregateVault.sol	90e57f19118bd3430b0e3bcb40084b7dabe903db
BV	BaseVault.sol	08d6e03ac0013a03fa457f144f45eb5762627872
BH	BaseHandler.sol	c1d85908fbc6118c2fdb217d56719076a43a9b13
GMI	GMI.sol	4a9d80d66e6fd30dbbdd3113664633b0eeaeeb1c
GMIH	GmiV2Handler.sol	480d456d3b0ab201658e72a2cd4ceb628066ebbf
GMIS	GmiStorage.sol	ecd097af7e96320926c9080cb13c505ade1018cb

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
GMIU	GmiUtils.sol	61bdadf9ff3ed1d22ba3e3baef768f14dd67a3fc
VF	VaultFees.sol	955d80ad93b5fe54a772d1f59d5daacfcf422b91
SGV	StorageViewer.sol	87d84e99bf6a113041ab3a7c2c284a0148bd924b
AGVH	AggregateVaultHelper.sol	ba4877395217fd4338907191596006ec40ae2327
GVH	GmxV2Handler.sol	6024434f514dc62df6a4d824351d540c7177aec0
EMIT	Emitter.sol	8359b41c6ee6fd421edb4bc1cf61fc51973102d5
RH	RequestHandler.sol	8936994e863b70e85e87f8ee0f52b1308dabcd3e
HH	HookHandler.sol	31a79f2528f8ba42b13189b852619eb75bae4cb0
PV	PausableVault.sol	ca5270ab1a184706f6c0620778795a0c14a53e08

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 Umami's GM Vaults, fuzz-testing with Foundry was performed on the protocol's main functions. 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+ runs up to a depth of 20 with a prepared Foundry fuzzing suite.

ID	Description	Tested	Passed	Run Count
AV-01	TVL of a Vault = PPS * Share Supply	✅	❌	-
AV-02	Sum of User Vault Share Balances Does Not Exceed Total Supply	✅	✅	10,000+
AV-03	Zero Address Vault Share Balance is Zero	✅	✅	10,000+
AV-04	Total supply of Asset Vault Not Modified Until Request Is Executed	✅	✅	10,000+
AV-05	Asset Vault Shares Decreased By Amount Requested on Redeem	✅	✅	10,000+
AV-06	TVL Does Not Increase After A Rebalance Without Price Change	✅	✅	10,000+
AV-07	Total Supply Increased After Deposit	✅	❌	-
AV-08	Assets Increased By Amount Deposited	✅	✅	10,000+
AV-09	Previewed Shares = Minted Shares Without Price Movement	✅	✅	10,000+
AV-10	Total Supply Decreased After Redeem	✅	✅	10,000+
AV-11	Shares Decreased By Amount Redeemed	✅	✅	10,000+

Invariants Assessed

ID	Description	Tested	Passed	Run Count
AV-12	Previewed Assets = Redeemed Assets Without Price Movement	✅	✅	10,000+
GLOBAL-01	Rebalance Brings Allocation Closer to Target than Original	✅	✅	10,000+
GMI-01	Sum of GMI Balances Of Both Vaults = GMI Total Supply	✅	❌	-
GMI-02	GMI Balance of Aggregate Vault Does Not Exceed GMI Total Supply	✅	✅	10,000+
GMI-03	Zero Address GMI Share Balance is Zero	✅	✅	10,000+

Findings & Resolutions

ID	Title	Category	Severity	Status
RH-1	All Asset Vault Funds Can Be Stolen Through Callbacks	Logical Error	Critical	Resolved
AV-1	Request Can Be Cancelled For Other Asset Vault	Validation	Critical	Resolved
AV-2	Stale LLO Prices DoS	DoS	Critical	Resolved
GVH-1	DOS Rebalance Through Simple Transfer	DoS	High	Resolved
GMIU-1	Entire Misallocation Covered On Deposit Or Withdrawal	Logical Error	High	Resolved
AGV-1	Mishandling Of Gas Stipends	Logical Error	High	Resolved
RH-2	Wrong Withdrawal Fee Calculation Parameters Passed	Logical Error	High	Resolved
VF-1	Wrong GMI Conversion Calculation	Logical Error	High	Resolved
LCY-1	Incorrect GMI Attribution	Logical Error	High	Resolved
RH-3	Wrong Gas Stipend Passed To Callback	Logical Error	Medium	Resolved
VF-2	Rebalance Fees Errantly Account For Withdrawals	Logical Error	Medium	Resolved
GMI-1	GMI Allocations Incorrectly Handle Saturated Markets	Logical Error	Medium	Resolved
GMI-2	Deposit Prevented By Double Counting GM Deposits	Logical Error	Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
RH-4	Lack Of Slippage On Deposits And Withdrawals	Slippage	Medium	Resolved
LCY-2	Zero Slippage Protection On Swaps	Logical Error	Medium	Acknowledged
AV-3	Vault Cap Can Be Bypassed	Logical Error	Medium	Resolved
RH-5	Not Subtracting Full Size On Withdrawal	Logical Error	Medium	Resolved
RH-6	Deposit Failures Unexpectedly Refund The Account	Logical Error	Low	Resolved
GMIU-2	adjustToBalance Unbalances Upon Withdrawal	Logical Error	Low	Acknowledged
GMIU-3	GMI previewMint Rounds GM Amounts Down	Rounding	Low	Resolved
GMI-3	GMI Deposit Amount Rounded Down	Rounding	Low	Resolved
VF-3	Leap Years Are Unaccounted For	Leap Years	Low	Resolved
AGVH-1	getVaultPPS Rounds In Favor Of Deposits	Rounding	Low	Resolved
LCY-3	Rebalance Functions Accessible Outside Of Rebalance Periods	Access Control	Low	Resolved
AV-4	CallbackHandler not assigned in setPeripheral	Superfluous Code	Low	Resolved
GMIU-4	Weight Cannot Be 0	Warning	Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
AVH-2	assetVault Shares Collateral Risk	Warning	Low	Acknowledged
RH-7	Lacking Event For setCallbackEnabled	Events	Low	Resolved
GLOBAL-1	Unused Functions	Superfluous Code	Low	Resolved
RH-8	Request Gas May Not Match The Gas Provided By The User	Logical Error	Low	Acknowledged
AGV-2	Performance Fees Errantly Measured	Logical Error	Low	Resolved
LCY-4	Unnecessary vaultIdx Variable	Optimization	Low	Resolved
AV-5	Request Creator May Not Cancel The Request	Unexpected Behavior	Low	Resolved
LCY-5	Superfluous assetToMintFrom Variable	Optimization	Low	Resolved
AGV-3	Changing Fee Recipient Should Be Done Only After A Rebalance	Improvement	Low	Acknowledged
GLOBAL-2	No Validation Against Trapped Fees	Validation	Low	Acknowledged
VF-4	Excessive GMX Withdrawal Fees	Logical Error	Low	Acknowledged
PV-1	Pausing Or Unpausing Spams Identical Events	Improvement	Low	Resolved
AGV-4	High Netting Threshold Can Block Rebalancing	Validation	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
RH-9	Unused Callback Gas Not Refunded To User	Logical Error	Low	Acknowledged
AGV-5	Epoch Delta Cleared Before Fees Are Calculated	Logical Error	Low	Resolved
AGV-6	Stale Vault Index Allocation Used	Logical Error	Low	Acknowledged
AV-6	ETH Not Returned On Request Cancelation	Logical Error	Low	Acknowledged
AGV-7	AggregateVault Can Be Completely Drained By Excessive Fees	Centralized Risk	Low	Acknowledged
AGV-8	Attacker can Prevent Closing of Rebalance	Warning	Low	Acknowledged
AGV-9	Changing Fee Percentage Should Be Done After Rebalance	Logical Error	Low	Acknowledged
AGV-10	CLOSE_REBALANCE_HOOK Is Called Before Rebalance Gets Closed	Logical Error	Low	Acknowledged
LCY-6	Rebalance DoS With Empty Deposit Amounts	DoS	Low	Resolved
VF-5	Precision Loss When Calculating Fees	Precision	Low	Acknowledged
GLOBAL-3	Redundant Code	Superfluous Code	Low	Resolved
LAVU-1	Sum Of Vault's GMI Less Than Total Supply	Precision	Low	Acknowledged
AGVH-2	TVL Not Equal To PPS Multiplied By Shares	Precision	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
RH-10	Total Supply Does Not Increase After Deposit	Precision	Low	Resolved
GVH-2	Missing Minimum Output Amount On GMX Operations	Logical Error	Low	Acknowledged
LCY-7	Unused Output Amount Leads to Skew	Logical Error	Low	Acknowledged
VF-6	Management Fees Deducted Based On Performance	Logical Error	Low	Acknowledged
LCY-8	State Does not Unwinded Properly on Failed Fulfillments	Logical Error	Low	Acknowledged
RH-11	Revert Bytes Gas Griefing	Gas Griefing	Low	Acknowledged
GLOBAL-4	Lacking onlyDelegateCall Modifier	Modifiers	Low	Acknowledged

RH-1 | All Asset Vault Funds Can Be Stolen Through Callbacks

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

Description PoC

The request that is currently being executed in `RequestHandler.executeRequest()` is cleared at the end of the function. This presents a critical problem as users can execute a deposit/withdraw request with a callback to an arbitrary address that they pass by using `assetVault.depositWithCallback()` or `assetVault.redeemWithCallback()`.

This callback will be executed before the request gets removed, leaving room for exploitation. `assetVault.cancelRequest()` immediately cancels a request and returns the funds to the user.

1. Create a deposit/withdraw request with a callback to an arbitrary contract we control.
2. The keeper picks up the request and executes it.
3. We call `assetVault.cancelRequest()` in the `afterDepositExecution()`/`afterWithdrawalExecution()` callback to cancel the request and return the funds to us immediately.
4. We now have the same funds/vault shares as before the request but have also received the funds from the request.

The exploit described above puts all funds in the asset vaults at risk of being stolen.

Recommendation

Call `aggregateVault.clearRequest(key)` before executing the callback.

Resolution

Umami Team: The issue was resolved in commit 8227df2

AV-1 | Request Can Be Cancelled For Other Asset Vault

Category	Severity	Location	Status
Validation	● Critical	AssetVault.sol: 146	Resolved

Description PoC

When a user cancels their request with the function `cancelRequest`, it is verified that the user cancelling the request is indeed the `sender` who sent the request. Afterwards, the funds contained in the Asset Vault are sent to the user depending on the amount of the request.

However, there is no validation done to ensure that a user who deposited/redeemed into one Asset Vault is not cancelling the created request on the other Vault.

For example, for illustrative purposes, consider the drastic scenario of a user creating a 1 ether deposit into the ETH Vault. The user can then trivially call `cancelRequest` on the USDC Vault, and be refunded 1e18 USDC. This leads to an enormous loss of funds for the USDC Vault depositors which is extremely easy to perform.

Recommendation

Use the `vault` attribute of the `OCRequest` to ensure that cancellation is only performed for the Asset Vault in which the deposit/redeem was intended.

Resolution

Umami Team: The issue was resolved in commit ea5e0e7

AV-2 | Stale LLO Prices DoS

Category	Severity	Location	Status
DoS	● Critical	AssetHandler.sol: 385, 403	Resolved

Description

Chainlink LLO prices are only updated during the opening and closing of rebalancing periods and the `RequestHandler.executeRequest` function. Yet, LLO prices are used in the logic for users to initiate withdraws/deposits on the `AssetVault` when previewing the deposit fee with the `previewDepositFee` function and previewing the withdrawal fee with the `previewWithdrawalFee` function.

Stale LLO prices, which users cannot update themselves, will cause users's withdraw/deposit initiations to revert as the withdraw/deposit fee estimation code checks that the LLO prices are no more stale than 1 Arbitrum block.

Additionally, protocol operators should update the latest LLO prices before calling the `cycle` or `fulfulRequests` functions during rebalance as these functions rely on up-to-date LLO prices.

Recommendation

Do not rely on LLO pricing for the user-initiated deposits and withdrawals. Instead in the `previewDepositFee` and `previewWithdrawalFee` functions pass `false` as the `useLlo` value.

Resolution

Umami Team: The issue was resolved in commit [54ada0e]

GVH-1 | DOS Rebalance Through Simple Transfer

Category	Severity	Location	Status
DoS	High	GmxV2Handler.sol: 148	Resolved

Description PoC

A Denial of Service (DoS) attack can occur on rebalances by forcing the Umami-calculated `estimateExecutionFee` to be less than GMX's `minExecutionFee`.

As only one token is being deposited on a rebalance, Umami calculates the `estimateExecutionFee` and enters an if statement, returning the smaller value compared to if it were to deposit two tokens.

The issue arises when an attacker forcibly sends 1 wei of the opposite token to GMX. The other gas limit value is then used, which is greater than what Umami used to calculate `estimateExecutionFee`.

This leads to a revert in the `validateExecutionFee` function. With this attack, it becomes impossible to perform a rebalance, rendering the core feature of the protocol unusable.

Recommendation

Send excess WETH for the execution fee, as any excess would be refunded anyway. This mitigates the risk of the DoS attack and ensures the rebalance functionality remains functional.

Resolution

Umami Team: The issue was resolved in commit 8f3f436

GMIU-1 | Entire Misallocation Covered On Deposit Or Withdrawal

Category	Severity	Location	Status
Logical Error	High	GmiUtils.sol: 72	Resolved

Description PoC

In the adjustToBalance function when the shareValue is insufficient to cover the entire underAllocation, the difference array with the entire positive underAllocations is returned. However, the shareValue is insufficient to cover these underAllocation amounts.

As a result, whenever a share amount is minted that is unable to cover the entire underAllocation, the entire underAllocation amounts will be charged to the caller while only remunerating the caller with the insufficient share amount that was specified.

This issue is most clearly demonstrated with a mint of a single wei. The single wei will be insufficient to cover the entire underAllocation, as a result, the caller is errantly required to provide the entire underAllocation amount in GM tokens to mint the specified single wei of GMI.

Similarly, this issue is present with withdrawals, where redeeming a single wei of shares will result in the withdrawer receiving the entire over-allocation amount.
This is a fundamental accounting error and will significantly affect the assetVault share values and the GMI valuation over time.

Recommendation

Replace the return statement on line 72 with:
Solarray.arrayAddProportion(toBalanceAmount, shareValue, difference, underAllocation, true);

Resolution

Umami Team: The issue was resolved in commit ea5e0e7

AGV-1 | Mishandling Of Gas Stipends

Category	Severity	Location	Status
Logical Error	High	AggregateVault.sol	Resolved

Description PoC

The protocol charges users an amount for gas since the protocol employs an asynchronous model where keepers pick the transactions up and execute them. The protocol checks whether the user has sent enough ETH in `msg.value` and if so adds their transaction to the uncompleted transaction queue.

The issue here comes in due to how those gas fees are handled. The following checks whether the user has sent enough funds to cover the gas to be expended by the keeper:
`require(msg.value >= gas, "AggregateVault: !gasRequirement");`

The issue with the above check is that it assumes that `gas` is in terms of ETH instead of in gas units as it is.

Given that the gas stipend for a request is within the 100,000 - 1,000,000 range the transaction's gas cost on the the user's side will be extremely low - less than a billionth of a cent since ETH is in 18 decimals. This will cause the protocol to lose funds in keeper gas fees on every deposit/withdrawal request that gets executed.

Recommendation

Consider converting the gas units into a notional value before checking whether the amount passed by the user is sufficient by multiplying it by `tx.gasprice`.

Resolution

Umami Team: The issue was resolved in commit 8227df2

RH-2 | Wrong Withdrawal Fee Calculation Parameters Passed

Category	Severity	Location	Status
Logical Error	High	RequestHandler.sol:87	Resolved

Description PoC

The protocol charges users fees on deposit and withdrawal. Those fees are based on the percentages set by the protocol and on the size in the asset vault's native token of the amount being deposited/withdrawn.

The issue here is due to a share size being passed to the `aggregateVault.previewWithdrawalFee` function even though the function that calculates the said fee - `VaultFees.getWithdrawalFee()` assumes it is a native token amount.

As the vault's TVL grows and more yield is gained through its strategies, each share will be worth more. However, this will not be represented when calculating the withdrawal fee as the calculations will think that the amount of shares passed in is the native token amount.

This directly impacts the protocol as the fees it will receive on withdrawal will be substantially lower than expected leading to a loss of fees for the protocol.

Recommendation

To mitigate the issue convert the vault shares into their native asset's worth before passing them to `aggregateVault.previewWithdrawalFee()`.

Resolution

Umami Team: The issue was resolved in commit ea5e0e7

VF-1 | Wrong GMI Conversion Calculation

Category	Severity	Location	Status
Logical Error	High	VaultFees.sol:137	Resolved

Description

GMX fees get added on top of the base withdraw/deposit fees if they are enabled through `shouldUseGmxFee`. In the case of a withdrawal, those fees are calculated based on the withdrawal size in GMI.

The issue arises due to the following line, which gets used to turn the withdrawal size into GMI, which then gets turned into corresponding GM token amounts: `gmi.sharesToMarketTokens(size * gmi.pps(prices) / 10 ** ERC20(asset).decimals(), prices)`.

The formula used for the calculation does not convert a USD notional amount into GMI, but quite the opposite.

This will always lead to a much larger fee due to the skewed GM token amounts, thus losing users' funds through excessive fees.

Recommendation

Convert `size` into a USD notional value and use a formula for converting USD into GMI: `size * 1e18 / gmi.pps(prices)`.

Resolution

Umami Team: The issue was resolved in commit [ea5e0e7]

LCY-1 | Incorrect GMI Attribution

Category	Severity	Location	Status
Logical Error	High	LibCycle.sol: 403-420	Resolved

Description PoC

The global state variable vaultGmiAttribution represents the exact amount of GMI attributed to each vault. Throughout the codebase it is repeatedly set using the _commitGmiDeltaProportions function from the LibCycle library.

This function incorrectly uses the vaultGmiAttribution values as percentages, which they are not, instead of absolute values. Because of this, the prevAmounts are extremely large and any _amt being added to will be minuscule in comparison which leads to almost 0% change in the proportions. The larger the amount is, the more the allocations will deviate from the intended values.

A direct, severe issue, appears during a rebalance when internally swapping GMI for native assets for internally settable differences between the 2 vaults. After swapping native tokens (L146-L152) the new allocations need to be saved from each vault, removing GMI from one vault and adding to another (L156-L172).

Since the call to _commitGmiDeltaProportions results in no practical change in the percentage, users are directly losing funds via depreciation of vault shares, since ETH will be swapped in these cases but GMI attribution has not changed.

To illustrate the impact, consider the following scenario:

- Total GMI valuation of $6 million
- Initial vault GMI allocation: 57% (57.0000090250015061%) USDC vault and 43% (42.9999909749984939%) WETH vault an amount approx $300K GMI (5% of GMI amount) is needed be moved from one vault to another

In this case, the current, incorrect implementation shows that the vault allocation, after adding the new amount, is: USDC: 57% (57.0000095000015852%), ETH: 43% (42.9999904999984147%). The new amount impact is erased and $300K worth of ETH is not GMI attributed.

If the correct implementation would be used, the resulting allocations are USDC: 60% (60.0000095000015853%), ETH: 40% (39.9999904999984146%). The error in this case is an absolute 3% value in allocation.

Recommendation

Modify the _commitGmiDeltaProportions to correctly work with and save the values as absolute amounts.

Resolution

Umami Team: The issue was resolved in commits 90b2627 and e3e3cef

RH-3 | Wrong Gas Stipend Passed To Callback

Category	Severity	Location	Status
Logical Error	Medium	RequestHandler.sol	Resolved

Description

The storage of `AggregateVault` has two variables for the two different gas fees that are paid by users when they create a request: `executionGasAmount` and `executionGasAmountCallback`. The former is for normal requests and the latter is for requests with callbacks.

The issue arises due to how `executionGasAmountCallback` is handled when calling the callback address the user provided. `executionGasAmountCallback` is passed directly as a gas stipend to the callback call even though it is intended to cover the whole call.

This issue causes the protocol, and more specifically the keeper, to provide a much higher gas stipend to the callback, resulting in loss of funds on every transaction. Another potential issue is the depletion of the keeper's ETH balance through large amounts of malicious requests aimed at disrupting deposits and withdrawals of innocent users.

Recommendation

Consider passing `executionGasAmountCallback - executionGasAmount` as a gas stipend to callback calls.

Resolution

Umami Team: The issue was resolved in commit [8227df2]

VF-2 | Rebalance Fees Errantly Account For Withdrawals

Category	Severity	Location	Status
Logical Error	Medium	VaultFees.sol: 228	Resolved

Description

When the rebalance fees are computed with the `_getVaultRebalanceFees` function, the magnitude of the funds withdrawn during the epoch is added to the `lockedBalanceSansUserDelta` with the `userPositionDelta` variable. The resulting `lockedBalanceSansUserDelta` is ultimately the amount that is fee'd.

However this incorrectly fees the remaining assets in the vault, the issue becomes clear considering the following (unreasonable, yet demonstrative) example:

- 90% of the funds in the vault are withdrawn in a single epoch
- 10% of the funds remain, and the users holding that remaining amount are subject to a fee based upon the entire 100%.
- Those who withdrew are not subject to this fee.
- The remaining users are exposed to an exorbitant fee as a percentage of their holdings.

This specific example is hyperbolic and unlikely to ever arise but is used merely to demonstrate the inaccuracy of the fee logic and the smaller-scale inequality that will occur on every rebalance.

Additionally, the current fee calculations clearly misaccount these withdrawn amounts because they are treated as if they were in the system for the entire epoch. The `performanceFeePercent`, `managementFeePercent`, and `timelockYieldPercent` are all computed based on the `percentYear` of the past epoch and applied to these withdrawn amounts.

However, the withdrawn amounts by definition cannot have been present in the vault for this entire period, in the worst case they will have been withdrawn from the vault at the beginning of the epoch.

Recommendation

Do not fee the remaining vault amounts based on the withdrawn amounts during the epoch.

Resolution

Umami Team: The issue was resolved in commit [ea5e0e7]

GMI-1 | GMI Allocations Incorrectly Handle Saturated Markets

Category	Severity	Location	Status
Logical Error	Medium	GMI.sol: 406	Resolved

Description

When a rebalance is underway, the GMI shares to be minted are validated by the `_validateMintableAmounts` function from the `GMI` contract.

The function incorrectly considers that the maximum allowed USD equivalent value to be deposited into the asset-specific GMX pool is via the backing token with the lowest availability, not the highest. This results in incorrect asset allocations for cases where the equivalent amount value cannot be deposited in the pool via the saturated backing token, but could have been deposited in the other one.

In `_validateMintableAmounts`, the maximum ETH value in USD (`mintableEth`) and maximum USDC value in USD (`mintableUsdc`) that can be deposited into each GMX asset market per backing token are calculated. Out of these two amounts, the largest should be selected as exactly how much can be deposited into the specific GMX pool using only one operation.

The issue is that the `maxMintable` chooses the smaller, not the larger out of the 2 values.This results in an incorrect maximum allocation amount for that particular asset pool, lower than it can be deposited. Consider a situation where a GMX pool gets long saturated and the protocol does a rebalance towards the short token.

The `_validateMintableAmounts` function will incorrectly indicate that the maximum you can deposit into that saturated pool is almost nothing since it uses the lowest available amount from the saturated one for validation.

This situation would result in depositing into the fallback pool, which will revert when also saturated. Ultimately, the protocol becomes imbalanced, risking the loss of user funds.

Recommendation

Base the `previewMint` and `_validateMintableAmounts` functions maximum mint amount on the asset being used to mint, not always take the greater or the smaller one. This would eliminate any issue that may appear due to over or underestimating the maximum amount.

Resolution

Umami Team: The issue was resolved in commit [3940624]

GMI-2 | Deposit Prevented By Double Counting GM Deposits

Category	Severity	Location	Status
Logical Error	Medium	GMI.sol: 90	Resolved

Description

In the `deposit` function, the `previewMint` function is called with the shares that are to be minted to the caller for their deposited GM amounts.

The `previewMint` function contains the `_validateMintableAmounts` validation at the end of the function which accounts for the share value being deposited into the GMX V2 system and reverts if the additional deposit tokens would put the GM market over the deposit cap.

However, during a deposit, these GM tokens have already been minted and there are no additional long or short tokens that will be deposited into the GM market.

Therefore, this validation erroneously accounts for long/short tokens being deposited when they will not be, and as a result, causes unnecessary reverts when these phantom long/short token amounts exceed the deposit cap in GMX V2, ultimately causing DoS attacks on deposits.

Recommendation

Do not perform the `_validateMintableAmounts` validation when depositing already minted GM tokens into GMI.

Resolution

Umami Team: The issue was resolved in commit ea5e0e7

RH-4 | Lack Of Slippage On Deposits And Withdrawals

Category	Severity	Location	Status
Slippage	Medium	RequestHandler.sol: 85-89, 97-100	Resolved

Description

Deposits and withdrawals to/from the asset vaults are a 2 step operation. The user initiates the action, thus creating a pending order and a keeper asynchronously executes that operation.

Although the execution keeper operates relatively fast, between 1-3 blocks since the initial request, there will still exist situations where an operation is initiated exactly before a rebalance is opened. During a rebalance, operations cannot be executed by the keeper, as such the user action will only be executed after the rebalance closes.

An issue is that users expect their deposit/redeem to result in the exact amounts indicated by the `previewDeposit` and `previewRedeem` functions at that time, but because of the price being recalculated again at the time the operation is executed, users may experience negative slippage and obtain fewer tokens.

During an epoch, a meaningful difference may not appear due to fast keeper response, but for those transactions that ultimately do become pending during a rebalance, the price difference may be significant enough of a loss. Since these operations flow normally, this situation will occur.

Recommendation

Add a slippage parameter when users deposit/redeem into the vault which will be passed and used by the `RequestHandler` when invoked by the keepers. Since the vaults are not meant to be ERC4626 compliant, this alteration does not come with a negative impact on the protocol.

Resolution

Umami Team: The issue was resolved in commit [ea5e0e7]

LCY-2 | Zero Slippage Protection On Swaps

Category	Severity	Location	Status
Logical Error	Medium	LibCycle.sol: 390	Acknowledged

Description

In the `rebalanceGmi` function, if `isOppositeDirection` is true, it attempts to swap one token for the other to rebalance before the minting and burning processes. This is achieved by swapping through UniswapV3.

When the swap is executed, the `_minOut` is set to zero. This means that regardless of how much slippage the swap incurs, the execution will continue. This poses a security risk, as attackers can perform a sandwich attack on this swap on UniswapV3 and steal funds if they have sufficient capital.

The impact of this is that any attempt to rebalance while `isOppositeDirection` is true will lead to an excess loss of funds due to the lack of slippage protection.

Recommendation

Set a `_minOut` value so that swaps do not incur more slippage than expected.

Resolution

Umami Team: Acknowledged.

AV-3 | Vault Cap Can Be Bypassed

Category	Severity	Location	Status
Logical Error	Medium	AssetVault.sol	Resolved

Description PoC

Before depositing funds into the protocol, the `deposit` functions perform a check to ensure the vault cap will not be exceeded:
`require(totalAssets() + assets <= previewVaultCap(), "AssetVault: over vault cap");`

However, the `totalAssets()` function does not consider the funds that are still pending to be sent into the `AggregateVault`. As a result, User A can deposit funds that reach the cap but will not be included in the TVL.

User B will then make another deposit, and since the current TVL has not been updated yet, their deposit to AssetVault will also go through. Once both requests are settled, the vault cap will be bypassed.

Recommendation

Validate the vault cap upon request execution so that it cannot be easily exceeded by a potentially significant amount.

Resolution

Umami Team: The issue was resolved in commit ea5e0e7

RH-5 | Not Subtracting Full Size On Withdrawal

Category	Severity	Location	Status
Logical Error	Medium	RequestHandler.sol:92	Resolved

Description PoC

`epochDelta` is used to track the amount of deposits/withdrawals during an epoch. It grows positive when more funds are being deposited than being withdrawn and vice versa.

The issue arises due to how the protocol increments and decrements `epochDelta`. The protocol increments the delta with the amount deposited after the fees get subtracted from it, thus only incrementing with the amount that entered the `AggregateVault`.

`aggregateVault.incrementEpochDelta(underlyingToken, assetsSansFees.toInt256())`
However, the same pattern is not followed in the withdrawal logic. Instead of decreasing the whole amount that leaves the vault only `size - fees` get subtracted.

`aggregateVault.incrementEpochDelta(underlyingToken, -(assetsSansFees.toInt256()))`
This will result in an imbalance where depositing the same amount has a higher impact on increasing `epochDelta` compared to the mitigating effect of withdrawing, thereby exposing the protocol to fund loss due to reduced fees.

This happens due to the subtraction of positive `epochDelta` from the current TVL during withdrawal fee calculations.

Recommendation

Decrement `epochDelta` by `assets` instead of `assetsSansFees` .

Resolution

Umami Team: The issue was resolved in commit 54ada0e

RH-6 | Deposit Failures Unexpectedly Refund The Account

Category	Severity	Location	Status
Logical Error	Low	RequestHandler.sol: 58	Resolved

Description

When a deposit request is made on behalf of another account and fails execution, the native assets are sent to the `receiver` instead of the `sender` of the request, who initiated the request and provided the funds.

This may be unexpected behavior as the sender would expect to receive their funds back if the request was not successfully executed.

Recommendation

Consider sending the funds to the `sender` on a deposit request fail.

Resolution

Umami Team: The issue was resolved in commit ea5e0e7

GMIU-2 | adjustToBalance Unbalances Upon Withdrawal

Category	Severity	Location	Status
Logical Error	Low	GmiUtils.sol: 85	Acknowledged

Description

In the adjustToBalance function, when there is an overallocation of GM tokens during a deposit the overallocation is offset by depositing into all other GM tokens at a ratio between the weight of the over-allocated market and the other market. This is desired during deposits as the higher-weighted market should receive a greater amount to move closer to the desired weightings.

However, during a deposit, the opposite occurs. A larger portion of GM tokens are withdrawn from the higher target weight market, ultimately moving the market balances further away from the desired weightings rather than closer.

For example, consider the following scenario:
The desired weightings are 0.2, 0.3, 0.25, and 0.25 respectively. And the respective GM token balances are 25, 25, 20, and 30.

The third market is underallocated by 5 GM tokens so markets 0,1, and 3 will have an increased withdrawal amount relative to the proportion of their target weighting to the target weighting of market 2.

Market 0 will only be decreased by 4 GM tokens, while market 1 will be decreased by 6 GM tokens. However the desired weighting of market 1 is higher than that of market 0, therefore this rebalancing moves the GM distribution further away from the desired weighting.

Withdrawals will often move GMI away from the desired weighting of GM tokens resulting in the desired positions not being met. This directly undermines the protocol's goal of maintaining a balanced index for GMI.

Recommendation

When the adjustToBalance function is being used in a withdrawal context the balanceWeightings should be calculated using a weights[i] / weights[j] ratio rather than the weights[j] / weights[i] ratio so that higher weighted GM markets are reduced less than lower weighted ones.

Resolution

Umami Team: Acknowledged.

GMIU-3 | GMI previewMint Rounds GM Amounts Down

Category	Severity	Location	Status
Rounding	Low	GMIUtils.sol: 68	Resolved

Description

In the `previewMint` function, the `gmValueToMint` returned from the `GmiUtils.adjustToBalance` function is rounded down as the `shareValue` which is used to determine the GM token amounts required uses round-down division.

Therefore the amount of GM tokens that the `aggregateVault` will use to mint GMI is often less than the value of the GMI shares that the `aggregateVault` receives.

This is not an issue when the `aggregateVault` holds all of the GMI shares, however in the future when other third-party actors may also hold GMI, then all other holders of GMI are diluted by deposits.

Recommendation

Consider rounding the `shareValue` up in the `adjustToBalance` function to avoid diluting other holders of GMI for the future where the `aggregateVault` is not the only holder of GMI.

Resolution

Umami Team: The issue was resolved in commit [54ada0e]

GMI-3 | GMI Deposit Amount Rounded Down

Category	Severity	Location	Status
Rounding	Low	GMI.sol	Resolved

Description

When depositing GMI, the computed PPS used to calculate the shares received is rounded down. However, in the future, GMI may be supported for other third-party users. In that case, it would be important to round the PPS up upon deposits.

This way, any third-party users would receive less share value than the amount of GM tokens they deposit, rather than more share value compared to the amount of GM tokens they deposit.
The precision loss due to rounding the PPS down is trivial but may pose a risk in the future.

Recommendation

Consider rounding the PPS up when computing the amount of shares to mint upon depositing into GMI.

Resolution

Umami Team: The issue was resolved in commit 54ada0e

VF-3 | Leap Years Are Unaccounted For

Category	Severity	Location	Status
Leap Years	Low	VaultFees.sol	Resolved

Description

In the `VaultFees` contract the YEAR constant is defined as being exactly 365 days in seconds, rather than 365.25 days to account for leap years.

Recommendation

Consider changing the YEAR value to `31557600` to account for leap years.

Resolution

Umami Team: The issue was resolved in commit [b919f11]

AGVH-1 | getVaultPPS Rounds In Favor Of Deposits

Category	Severity	Location	Status
Rounding	Low	AggregateVaultHelper.sol: 105, 111	Resolved

Description

In the `getVaultPPS` function, round-down division is used regardless of whether the function is being used to determine the shares a user receives upon a deposit or the assets a user receives on withdrawal.

This behavior rounds in the favor of the user upon deposits, allowing the user to deposit at a lower PPS due to precision loss. Though the precision loss is minor and is unlikely to have an impact it could be leveraged in a more complex attack.

Recommendation

To avoid any potential manipulations as a result of this precision loss, consider rounding up when the `getVaultPPS` function is being used to determine the amount of shares a user will receive for a deposit and rounding down when the `getVaultPPS` function is being used to determine the amount of assets a user will receive from a withdrawal.

Resolution

Umami Team: The issue was resolved in commit 54ada0e

LCY-3 | Rebalance Functions Accessible Outside Of Rebalance Periods

Category	Severity	Location	Status
Access Control	Low	LibCycle.sol	Resolved

Description

Neither the `cycle` function nor the `fulfilRequests` function validate that the system is indeed within a rebalancing period when they are being called.

These functions can only be triggered by a trusted party, however, they should be restricted to only rebalance periods as important validation and caching must occur before these functions are invoked.

Recommendation

Validate that the protocol is currently in a rebalancing period in the `cycle` and `fulfilRequests` functions.

Resolution

Umami Team: The issue was resolved in commit [b919f11]

AV-4 | CallbackHandler not assigned in setPeripheral

Category	Severity	Location	Status
Superfluous Code	Low	AggregateVault.sol	Resolved

Description

The `setPeripheral` function does not allow a value to be assigned for the `Peripheral.CallbackHandler`. Nor is the `Peripheral.CallbackHandler` is used anywhere in the codebase.

Recommendation

Consider removing the redundant `Peripheral.CallbackHandler` value from the `Peripheral` enum, otherwise implement the desired use case for the `Peripheral.CallbackHandler` value.

Resolution

Umami Team: The issue was resolved in commit [b919f11]

GMIU-4 | Weight Cannot Be 0

Category	Severity	Location	Status
Warning	Low	GmiUtils.sol: 85	Acknowledged

Description

When distributing the overallocated difference, the algorithm divides the `jth` weight by the `ith` weight. If the ith weight is 0, a division by 0 revert will occur.

Because `adjustToBalance` is called in `previewMint` which is used in function `_increaseGMI`, a rebalance could fail due to a reverted cycle operation.

Recommendation

Ensure none of the weights are set to 0.

Resolution

Umami Team: Acknowledged.

AVH-2 | assetVault Shares Collateral Risk

Category	Severity	Location	Status
Warning	Low	AggregatorVaultHelper.sol: 92	Acknowledged

Description

The `assetVault` shares are valued at the cached `vaultState.rebalancePPS` during the rebalance period, but as soon as the rebalance period is closed, the valuation of the `assetVault` shares will experience a stepwise jump to the current valuation.

Because of this stepwise jump, the `assetVault` shares should not be a candidate for collateral in any borrow/lending system, as there is a risk that a malicious user could deposit vault shares as collateral while they are valued at the cached price and ultimately have an insolvent position when the `assetVault` share price drops.

Borrow/lending platforms introduce a safety threshold between liquidation and solvency to address this, however there is a small risk that the magnitude of the stepwise jump exceeds this safety threshold. Any occurrence of this should be rare as rebalances are not intended to be large.

Recommendation

This is simply a warning to anyone who would integrate with the Umami GMI system and potentially accept the `assetVault` shares as collateral.

Resolution

Umami Team: Acknowledged.

RH-7 | Lacking Event For setCallbackEnabled

Category	Severity	Location	Status
Events	Low	RequestHandler.sol: 37-39	Resolved

Description

When the callback functionality is enabled by the `RequestHandler` contract, by calling the `setCallbackEnabled` function, there is no event is emitted.

Recommendation

Emit an event on the call to the `setCallbackEnabled` function.

Resolution

Umami Team: The issue was resolved in commit 54ada0e

GLOBAL-1 | Unused Functions

Category	Severity	Location	Status
Superfluous Code	Low	Global	Resolved

Description

The following functions are not used in the protocol:

- `_vaultGmiProportion`
- `_exactOutputSwap`

Recommendation

Consider removing these functions.

Resolution

Umami Team: The issue was resolved in commit [b919f11]

RH-8 | Request Gas May Not Match The Gas Provided By The User

Category	Severity	Location	Status
Logical Error	Low	RequestHandler.sol	Acknowledged

Description

The gas amount forwarded to the `request.callback` contract is always the current `executionGasAmountCallback` in storage:
`gas: storageViewer.getExecutionGasAmountCallback()`

This presents an issue as users are made to pay it on submitting the request:
`uint256 gas = _gasRequirement(callback != address(0));`

In the case of a change of the `executionGasAmountCallback` amount, users will either get more gas than they paid for or will get less.

Recommendation

Consider keeping the gas amount the user paid for as a value in the request struct and using it instead.

Resolution

Umami Team: Acknowledged.

AGV-2 | Performance Fees Errantly Measured

Category	Severity	Location	Status
Logical Error	Low	AggregateVault.sol: 228, 237	Resolved

Description

In the `closeRebalancePeriod` function, the performance fees for the previous epoch are collected after setting the `vaultState.rebalanceOpen` to `false`. As a result, the cached `vaultState.rebalancePPS` is not returned when using the `getVaultPPS` function to determine whether the performance fees should be charged.

This behavior errantly treats the current vaultPPS after the rebalance as if it applied for the entire period before the rebalance occurred.

This results in cases where the performance fee is missed due to the rebalance dipping the vaultPPS below the watermark.

This may occur as a result of swapping fees and/or fees from GMX. Additionally, the performance fee may be charged when instead it should not if the `vaultPPS` is increased above the watermark after the rebalance, though this case is rarer.

Recommendation

Close the rebalance period after fees are calculated so that the cached vaultPPS is used rather than the PPS resulting from the rebalance.

Resolution

Umami Team: Acknowledged.

LCY-4 | Unnecessary vaultIdx Variable

Category	Severity	Location	Status
Optimization	Low	LibCycle.sol: 191	Resolved

Description

In the `rebalanceGmi` function the `vaultIdx` is unnecessarily fetched using the `getTokenToAssetVauldIndex` function when the vault index is directly available as the index of the for-loop, `i`.

Recommendation

Use the for-loop index to indicate the vault rather than fetching the `vaultIdx` with the `getTokenToAssetVaultIndex` function.

Resolution

Umami Team: The issue was resolved in commit [b919f11]

AV-5 | Request Creator May Not Cancel The Request

Category	Severity	Location	Status
Unexpected Behavior	Low	AssetVault.sol: 84, 141, 149	Resolved

Description

In the `redeem` and `redeemWithCallback` functions, the owner is assigned as the `request.sender` when the request is created. The `request.sender` is used to validate the account that may cancel the request in the Umami system.

However, in some cases, the owner may not be the creator of the request. The owner may approve a third-party actor to create requests on their behalf. In this case, it would be appropriate to allow this third-party creator to cancel the request as well, however, they are not able to do so.

Recommendation

Consider changing the `request.sender` to the `msg.sender` in the `redeem` and `redeemWithCallback` functions. Otherwise, consider allowing both the owner and the request creator to cancel the request.

Resolution

Umami Team: The issue was resolved in commit [b919f11]

LCY-5 | Superfluous assetToMintFrom Variable

Category	Severity	Location	Status
Optimization	Low	LibCycle.sol: 210	Resolved

Description

In the `_increaseGmi` function the `assetToMintFrom` is assigned to the `_asset` parameter in every iteration of the for-loop, however, it is unnecessary to declare this `assetToMintFrom` variable as it will always be the `_asset` value.

Recommendation

Remove the `assetToMintFrom` variable declaration and use the `_asset` parameter value directly.

Resolution

Umami Team: The issue was resolved in commit [b919f11]

AGV-3 | Changing Fee Recipient Should Be Done Only After A Rebalance

Category	Severity	Location	Status
Improvement	Low	AggregateVault.sol: 532	Acknowledged

Description

Rebalancing the vaults results in a fee being paid to the designated `feeRecipient` at that time. This recipient, as it is, can be changed at any time by a configurator by calling the `setFeeRecipient` function from the `AggregateVault` contract.

Changing the fee at a random point represents a loss for the original fee recipient, as they have been the recipient since the last rebalance up until that point. Consider the situation during a normal operation period and right before opening a rebalance period.

Recommendation

When setting the new fee recipient, set a pending recipient, which is changed only when the rebalancing period is closed and the fee is paid. At that point, the fee is paid to the old recipient and the pending recipient becomes the actual fee recipient.

Resolution

Umami Team: Acknowledged.

GLOBAL-2 | No Validation Against Trapped Fees

Category	Severity	Location	Status
Validation	Low	Global	Acknowledged

Description

When configuring the `vaultFees` with the `setVaultFees` function, there is no requirement that the `depositFeeEscrow` and `withdrawalFeeEscrow` are configured to nonzero addresses if the deposit and withdrawal fees are configured to nonzero amounts.

Therefore, these fee amounts can get stuck in the vault contracts rather than being returned to the user or sent to any fee receiver.

Recommendation

Consider adding validation such that a nonzero fee amount cannot be configured without assigning the appropriate fee receiver to a nonzero address.

Resolution

Umami Team: Acknowledged.

VF-4 | Excessive GMX Withdrawal Fees

Category	Severity	Location	Status
Logical Error	Low	VaultFees.sol 137	Acknowledged

Description PoC

GMX swap fees get calculated and subtracted from a user whenever they deposit/withdraw shares from an asset vault. GMX fees in `VaultFees.getWithdrawalFee()` are calculated with the size's backing in GM tokens based on their respective weights.

The PPS and TVL of a vault are calculated with the liquid reserves of the native token in the aggregate vault, the GMI attributed to that vault, and also the opened external hedging position.

The issue here arises due to `VaultFees:137` assuming that the whole withdrawal size is backed in GMI tokens and calculating it accordingly. This makes users get charged a GMX fee on 100% of their withdrawal size instead of only the fraction that needs to be liquidated through GMX, thus making the users lose funds due to the excessive fees.

Recommendation

Consider calculating the GM market token amounts based on a fraction of `size` that corresponds to the `current fraction of GMI reserves / TVL`.

Resolution

Umami Team: This is intended. We charge the fee for the entire withdrawal as if it were coming from GMI.

PV-1 | Pausing Or Unpausing Spams Identical Events

Category	Severity	Location	Status
Improvement	Low	PausableVault.sol: 131-139, 148-156	Resolved

Description

When calling the functions `_pause` or `_unpause` from the `PausableVault` contract, for each function there will be between 1 and 3 identical events being emitted. This can cause confusion for any 3rd party integrator.

- for the `_pause` function: minimum 1 `Paused` event and maximum 2 more from the `_pauseDeposit` and `_pauseWithdrawal` function calls
- for the `_unpause` function: minimum 1 `Unpaused` event and maximum 2 more from the `_unpauseDeposit` and `_unpauseWithdrawal` function calls

Recommendation

For both the `_pause` and `_unpause` functions, remove the default emitted event, and for each pausing/unpausing event, either add an argument to identify if it was a deposit or withdrawal that was paused/unpaused, or create 2 separate events e.g. `PausedDeposits/PausedWithdrawals`.

Resolution

Umami Team: The issue was resolved in commit 54ada0e

AGV-4 | High Netting Threshold Can Block Rebalancing

Category	Severity	Location	Status
Validation	Low	AggregateVault.sol: 470-476	Resolved

Description

The `nettedThreshold` variable can be set by the configurator by calling the function `setThresholds` from the `AggregateVault` vault to any value.

Setting it to over the maximum BPS will result in blocking all rebalances that opt to validate netting due to an underflow operation in the `NettingMath` library at line 95.

Recommendation

Add a limit check for the `_newNettedThreshold` input in the `setThresholds` function so that it does not surpass the maximum BPS.

Resolution

Umami Team: The issue was resolved in commit [54ada0e]

RH-9 | Unused Callback Gas Not Refunded To User

Category	Severity	Location	Status
Logical Error	Low	RequestHandler.sol	Acknowledged

Description

Users can perform a deposit/withdrawal request with a callback to an arbitrary contract upon the completion of the request processing.

The issue arises because users are required to incur costs for the maximum gas amount they can utilize in the callback, even if their callback necessitates only a fraction of that for execution.

However, the unutilized gas in this process is not refunded, ultimately leading users to lose the value of any leftover gas in the callback.

Recommendation

Consider refunding the value of the gas units left at the gas price during the creation of the request.

Resolution

Umami Team: This is by design as the the keeper also has rebalance cost.

AGV-5 | Epoch Delta Cleared Before Fees Are Calculated

Category	Severity	Location	Status
Logical Error	Low	AggregateVault.sol: 220-221	Resolved

Description

The rebalancing fee is calculated based on the total TVL that was managed at the start of the current epoch. This fee is deducted when closing an epoch and, by calling the `closeRebalancePeriod` function from the `AggregateVault` contract.

The fee is incorrectly calculated because, in the `closeRebalancePeriod` function call, the epoch delta is cleared by a `_resetEpochDeltas` function call on line 221, before the actual fee collection is done on line 237. This results in the overall fee being calculated on the ending epoch TVL instead of the opening TVL.

A higher fee will be deducted if the epoch has more deposits and a lower fee will be deducted if the epoch has more withdrawals.

The end user is unaware of the fee value until the end of the epoch which, depending on the value, might have determined whether or not the user participated in the protocol during that epoch. Fee predictability is needed for users of the protocol.

Recommendation

Reset the epoch delta after the fees have been collected.

Resolution

Umami Team: The issue was resolved in commit [8227df2]

AGV-6 | Stale Vault Index Allocation Used

Category	Severity	Location	Status
Logical Error	Low	AggregateVault.sol:215,229	Acknowledged

Description

When closing the rebalance period, a validation is done so that the current vault holding ratio is within the accepted upper and lower thresholds.

This validation is incorrect because it uses the index allocations that were saved when the rebalancing period was opened, which are stale, instead of using the current index allocations determined at the moment the rebalance is closing, and that are saved in the `vaultIndexAllocation` variable.

Because of this, the vault ratio is always slightly incorrect since it calculates the vault index exposure as a factor of the current vault holdings (`vaultCumulativeHoldings`) but compares it to the vault holdings at the time when rebalancing was opened (`vaultHoldings`) in the function `vaultDeltaAdjustment` from the `NettingMath` library.

This slight difference may lead to the vault ratio exceeding the threshold and resulting in an execution revert. Or passing a ratio when it would exceed the threshold normally.

Recommendation

When closing the rebalance period, always use the current index allocation, not the stale one saved when the rebalance period was opened.

Resolution

Umami Team: We only care about the total GMI in the vaults when validating the netting check.

AV-6 | ETH Not Returned On Request Cancelation

Category	Severity	Location	Status
Logical Error	Low	AssetVault.sol: 146	Acknowledged

Description

When a user creates a request to either withdraw or deposit into Umami, they need to supply enough ETH to cover the execution fee.

This is so that the Umami keepers have funds to execute the transaction. Users who made the deposit or withdrawal can also cancel their request if it has not been executed yet.

The issue is that the ETH that the user supplied is not returned to the user. This is inconsistent behavior as the user loses any ETH that they sent for a request that never got executed.

Recommendation

Consider returning the ETH to the user since the keeper never used it.

Resolution

Umami Team: We chose to keep the gas here.

AGV-7 | AggregateVault Can Be Completely Drained By Excessive Fees

Category	Severity	Location	Status
Centralized Risk	Low	AggregateVault.sol: 415-425	Acknowledged

Description

When the vault fees are set by calling the function `setVaultFees` from the `AggregateVault` contract, there are no checks that do not surpass the equivalent of 100%.

A mistake by the configurator or a compromised configurator can set the fees to such a value that they equate the entire vault assets. When a rebalancing happens, the vault assets will be sent to the fee recipient, which can also be set by the configurator via `setFeeRecipient`.

Recommendation

Add limitations so that fees cannot surpass 100% but should also include a lower, maximum allowed threshold.

Resolution

Umami Team: Acknowledged.

AGV-8 | Attacker can Prevent Closing of Rebalance

Category	Severity	Location	Status
Warning	Low	AggregateVault.sol: 213	Acknowledged

Description

For the `closeRebalancePeriod` function to successfully execute, it must pass the check in the `checkNettingConstraint` function.

If the vault receives too little or too many GM tokens on a mint, it will cause the `vaultIndexAllocation` to change and push the `vaultRatio` outside of its bounds.

An attacker can take advantage of this by force-sending some ETH or USDC to the GMX `depositVault` while Umami is minting. This will cause Umami to receive more GM tokens than expected, which will increase the `vaultIndexAllocation` and consequently the `vaultRatio`.

If the vault ratio is already near the upper bound, it would only take a small amount to push the `vaultRatio` beyond the upper bound.

If the `closeRebalancePeriod` function cannot successfully execute, the protocol will remain in a rebalance state for longer than intended.

Recommendation

This manipulation would be capital-intensive for an attacker and can be avoided by disabling the netting validation. However, it would be prudent to be aware of this risk when setting targets that would put the `vaultRatio` near the upper limit.

Resolution

Umami Team: Acknowledged.

AGV-9 | Changing Fee Percentage Should Be Done After Rebalance

Category	Severity	Location	Status
Logical Error	Low	AggregateVault.sol: 415-425	Acknowledged

Description

Rebalancing the vaults results in a fee being paid in assets taken from the vaults themselves. Settings of the rebalance fee percentages can be directly changed by calling the `setVaultFees` function from the `AggregateVault` contract.

This is an issue, as changing the fee settings during an epoch alters the perceived risk that users associate when interacting with the vaults, high taxes would also make the vaults less appealing and in return would result in a lower utilization.

The `epochDelta` component as well as fee watermark PPS can be changed at any time by calling the `setAssetVaults` function from the `AggregateVault` contract. Again this is an issue as it changes the fee during an epoch.

Recommendation

When calling `setVaultFees`, have the new fee percentages be set in a pending state. When a rebalancing is executed, after fees are deduced using the old fee values, then change them to the pending ones.

Consider creating a separate function to update the fee watermark PPS value and date in the same manner as described above.

The function `setAssetVaults` should only be called on severe vault changes, consider limiting its use as much as possible.

Resolution

Umami Team: Acknowledged.

AGV-10 | CLOSE_REBALANCE_HOOK Is Called Before Rebalance Gets Closed

Category	Severity	Location	Status
Logical Error	Low	AggregateVault.sol	Acknowledged

Description

The protocol has two different types of hooks:

- Hooks calling arbitrary addresses passed by users after a request of theirs gets executed/canceled.
- Protocol hooks that get evoked when a request gets queued or a rebalance gets opened/closed.

`CLOSE_REBALANCE_HOOK` is called before deposits are unpaused, which may restrict the callback from certain operations and limit its potential behavior.

Recommendation

Consider calling the protocol hook after the rebalance period gets closed similar to how it gets called in `openRebalancePeriod`, just before the rebalance gets opened.

Resolution

Umami Team: Acknowledged.

LCY-6 | Rebalance DoS With Empty Deposit Amounts

Category	Severity	Location	Status
DoS	Low	LibCycle.sol: 213	Resolved

Description

Before depositing on GMX, the amount of asset vault native asset to expend to acquire the required amount of GM is calculated: `uint256 assetAmountRequired = _previewGmMint(markets[i], gmSharesRequired[i], assetToMintFrom);`

The `assetAmountRequired` can possibly be zero (post-internal netting), resulting in a GMX revert upon deposit creation with `EmptyDepositAmounts`. Because the mints are done in a for-loop, a failure in one market will cause all others to fail. Consequently, the `cycle` will fail and the necessary GMI shares will not be minted.

Recommendation

Carefully select the weights and target allocations such that function `_previewGmMint` does not return a zero amount.

Resolution

Umami Team: The issue was resolved in commit c07078c and 4467ccd.

VF-5 | Precision Loss When Calculating Fees

Category	Severity	Location	Status
Precision	Low	VaultFees.sol: 275, 284	Acknowledged

Description

When calculating the performance and management fee, division occurs before multiplication, leading to some precision loss, which makes the fee less than it should be.

Recommendation

Perform all division after all multiplication.

Resolution

Umami Team: Acknowledged.

GLOBAL-3 | Redundant Code

Category	Severity	Location	Status
Superfluous Code	Low	Global	Resolved

Description

Both `AggregateVaultHelper::_getVaultGmi` and `LibAggregateVaultUtils::getVaultGmi` implement the same functionality, with the only difference being how they fetch from storage.

Recommendation

Reuse the same functionality to avoid duplicative code.

Resolution

Umami Team: The issue was resolved in commit ea5e0e7

LAVU-1 | Sum Of Vault's GMI Less Than Total Supply

Category	Severity	Location	Status
Precision	Low	LibAggregateVaultUtils.sol: 84	Acknowledged

Description

It is possible for the sum of the two asset vaults' GMI allocations to be less than the total supply of GMI. As a result, a small portion of GMI is unaccounted for in the Asset Vaults which may lead to issues such as needing to expend more asset funds to reach the target allocation.

Recommendation

Clearly document this precision loss.

Resolution

Umami Team: Acknowledged.

AGVH-2 | TVL Not Equal To PPS Multiplied By Shares

Category	Severity	Location	Status
Precision	Low	AggregateVaultHelper.sol: 89	Resolved

Description

It is possible for the `TVL Of a Vault != PPS * totalSupply` because of precision loss when calculating the price per share. This may lead to slight differences in the amounts withdrawn and the shares received for a deposit, although this is preferable compared to having the `PPS * totalSupply` potentially exceeding the TVL.

Recommendation

Clearly document this precision loss.

Resolution

Umami Team: The issue was resolved in commit [b919f11]

RH-10 | Total Supply Does Not Increase After Deposit

Category	Severity	Location	Status
Precision	Low	RequestHandler.sol: 100	Resolved

Description

During a deposit it is checked that the shares to be minted is not 0: `require((shares = previewDeposit(assets)) != 0, "ZERO_SHARES");`

However, it is still possible for a user to mint 0 shares with a non-zero deposit due to the precision loss that occurs in `uint256 shares = assetsSansFees * (10 ** decimals) / pps;`
Consequently, a user may deposit a small amount of funds but receive 0 shares.

Recommendation

Consider enforcing a minimum deposit amount and/or validating the request upon execution.

Resolution

Umami Team: The issue was resolved in commit 54ada0e

GVH-2 | Missing Minimum Output Amount On GMX Operations

Category	Severity	Location	Status
Logical Error	Low	GmxV2Handler.sol: 137, 201, 202	Acknowledged

Description

In both the `mintGmTokens` and `burnGmTokens` functions, a minimum output amount is set to 0. This means that regardless of how many tokens are returned on a mint or burn, the execution will be completed successfully.

However, this poses a problem as any amount lost will negatively affect the vault's TVL, which would, in turn, impact the value of the users' shares.

Although some value loss is inevitable when making deposits or withdrawals on GMX v2, precautions should be taken to limit how much value can be lost.

This is especially important considering that the price impact can take out an unexpected amount of funds, and without a defined minimum output amount, price impact can remove a significant amount of value from the mint or burn, leading to a loss of funds for the users.

Recommendation

Use a non-zero minimum output amount for both minting and burning to limit the users' loss.

Resolution

Umami Team: Acknowledged.

LCY-7 | Unused Output Amount Leads to Skew

Category	Severity	Location	Status
Logical Error	Low	LibCycle.sol 385	Acknowledged

Description

In the `rebalanceGmi` function, if `isOppositeDirection` is true, it attempts to swap one token for the other to rebalance before the minting and burning processes. This is achieved by swapping whichever token is at a surplus for the other.

Any swap will experience slippage resulting in the output amount being different than the input amount. The issue is that when updating the `_current[]` as well as calling the function `_commitGmiDeltaProportions` for each of USDC and ETH, only `internalNet` is used.

This will lead to a skew in the accounting because the actual delta of the output token will be different than that of the input token.

This skew will lead to the wrong amount being minted/burned as the `cycle` function continues, as well as an undervaluing of whichever asset was the input token at the expense of the other token.

Recommendation

Change the state of the outputted token based on the returned output amount of the swap, instead of the inputted amount. This will align the actual token balance proportions with what is stored in the state, preventing misvaluing of assets and inaccurate minting/burning.

Resolution

Umami Team: We do this so the vault receiving the swap is responsible for all fees incurred from the swap (slippage and trading fee).

VF-6 | Management Fees Deducted Based On Performance

Category	Severity	Location	Status
Logical Error	Low	VaultFees.sol: 267-290	Acknowledged

Description

When calculating the withdrawal fee, the management fee component is incorrectly deducted only if the current vault price per share (PPS) is higher than the watermark PPS.

This condition is required for the performance fee, not for the management fee. This behavior is inconsistent with the way the management fee is deducted when rebalancing and leads to fewer fees for the protocol overall.

Recommendation

Calculate the management fee where there is a profit, regardless of the current vault price per share.

Resolution

Umami Team: We take the performance and management fee on regular intervals at rebalance time. On withdrawal we only take them if there has been a profit since we last took them.

LCY-8 | State Does Not Unwind Properly On Failed Fulfillments

Category	Severity	Location	Status
Logical Error	Low	LibCycle.sol: 318	Acknowledged

Description

When a rebalance occurs, and the protocol intends to mint GM tokens, it will deposit either WETH or USDC into GMX via the `mintGmTokens` function.

After the deposit is created, the GMX Keeper will execute the order, and on completion, send the GM tokens back to the Umami protocol. If the order is canceled or fails on execution for any reason, the GMX keeper will return the long or short token to Umami.

Because the function `_fulfilMintRequest` will loop through all of the `_mintRequest`, if one did not succeed then the whole transaction will revert due to the following require statement `if (!depositRequestDetails.success) revert RequestNotSucceeded();`.

Anytime the `depositExecution`on GMX v2 fails or anytime an order is canceled, those funds will require manual intervention. There are a variety of reasons both maliciously and unintended that can cause an order to fail upon execution.

For example, the max deposit cap can be exceeded during execution and not creation, as well as congestion on the network leading to the execution of the order taking place beyond the max block limit.

Recommendation

Carefully monitor the status of a rebalance and fix state inconsistencies with handlers when necessary.

Resolution

Umami Team: The keeper runs a simulation before sending all the transactions on chain but the possibility of them still failing after simulations is always there because of the state changes on chain between simulation block and rebalance block. If any of the requests fail it'd require manual intervention.

RH-11 | Revert Bytes Gas Griefing

Category	Severity	Location	Status
Gas Griefing	Low	RequestHandler.sol: 116, 128	Acknowledged

Description

In the `afterDepositExecution` and `afterWithdrawalExecution` functions in the catch block, arbitrary bytes are loaded into memory from the arbitrary request callback.

Although the callback is limited in gas expenditure by the `executionGasAmountCallback`, the arbitrary callback contract can cause the keeper to expend much more gas than expected by reverting with a large amount of revert bytes which are then subsequently loaded into memory.

Memory expansion costs a quadratic amount of gas and malicious revert bytes can lead to the keeper expending hundreds of thousands or even millions of additional unexpected gas units. Such an expenditure can cost the keepers a significant amount over a period of time.

Recommendation

Do not accept revert bytes from the callback contract.

Resolution

Umami Team: The eth_call is going to fail and keeper will continue on with the next one without actually sending any transaction and causing any loss of funds. The `gas` field for the transaction will be set to the base gas units `executionGasAmount` + callback gas units `executionGasAmountCallback` which will make it go OOG.

GLOBAL-4 | Lacking onlyDelegateCall Modifier

Category	Severity	Location	Status
Modifiers	Low	Global	Acknowledged

Description

There are several inconsistencies with contracts that have the functionality needed to be called on its own or delegate-called into. Functions that work only when delegate called-into require the `onlyDelegateCall` modifier will not return an invalid result when called into.

- For the `GmxV2Handler` contract:
  - `getDepositRequestDetails` and `getWithdrawRequestDetails` need the `onlyDelegateCall` modifier

- For the `VaultFees` contract:
  - `getDepositFee`, `getVaultRebalanceFees`, and `getWithdrawalFee` need the `onlyDelegateCall` modifier
  - consider making the following public functions internal, since they are called only from within the contract and also work only when delegate-called into: `_getVaultDepositFee`, `_getVaultWithdrawalFee`, `getVaultTVL`, and `getVaultPPS`

- For the `AggregateVaultHelper` contract:
  - `getTotalNotional` needs the `onlyDelegateCall` modifier
  - all functions from the `AggregateVaultViews` contract require the `onlyDelegateCall` modifier but with the sole exception of `vaultToAssetVaultIndex`, all other functions the not used by the `AggregateVaultHelper` contract.

Recommendation

Add an `onlyDelegateCall` modifier to the mentioned functions and implement the other suggested changes. Consider making `vaultToAssetVaultIndex` an internal function in the `AggregateVaultHelper` contract and removing the `AggregateVaultViews` contract completely.

If `AggregateVaultViews` is to be kept for on-chain reading of values through multicall, the `onlyDelegateCall` modifier must be added to all of its functions.

Resolution

Umami Team: Acknowledged.

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