UMAMI GLP VAULTS
SMART CONTRACT AUDIT

Report 1 of 2

April 19th 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

TECHNICAL SUMMARY

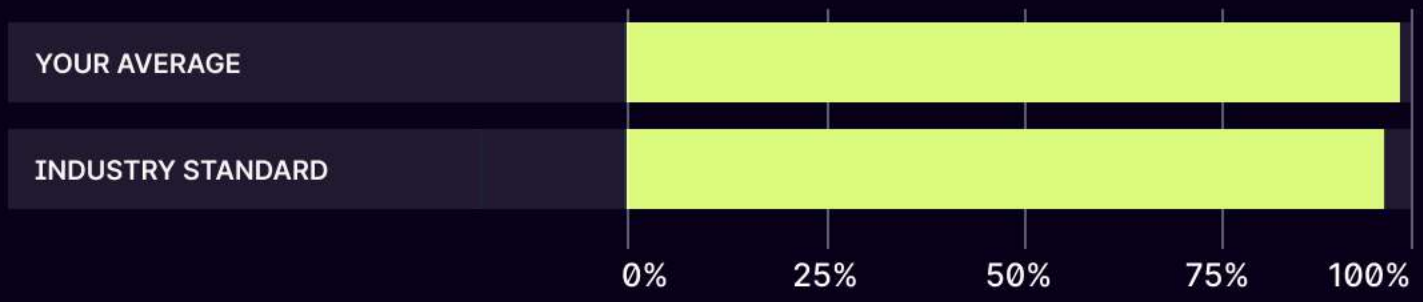
This document outlines the overall security of the Umami GLP Vaults smart contracts evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Umami GLP Vaults smart contracts codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the audit. (See Complete Analysis)

Testable Code



98.34% of the code is testable, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contracts but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Umami team put in place a bug bounty program to encourage further active analysis of the smart contracts.

Table of Contents

Auditing Strategy and Techniques Applied 3

Executive Summary 5

Structure and Organization of the Document 6

Complete Analysis 7

Code Coverage and Test Results for all files written by Zokyo Security 39

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Umami GLP Vaults repository: https://github.com/UmamiDAO/V2-Vaults.git

Last commit: 20bf713277508fd4cf42a900fc90cb8c3dfd15e7

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- BaseHandler.sol
- GlpHandler.sol
- BasePositionManager.sol
- GmxPositionManager.sol
- PositionManagerRouter.sol
- BaseSwapManager.sol
- GmxSwapManager.sol
- OneInchSwapManager.sol
- GlpPricing.sol
- NettingMath.sol
- ShareMath.sol
- SwapLibrary.sol
- VaultLifecycle.sol
- VaultStorage.sol
- Multicall.sol
- PositionMath.sol
- Solarray.sol
- TimeoutChecker.sol
- VaultMath.sol
- BaseWrapper.sol
- ChainlinkWrapper.sol
- UmamiPriceFeed.sol
- GlpRebalanceRouter.sol
- NettedPositionTracker.sol
- VaultFeeManager.sol
- AggregateVaultStorage.sol
- AggregateVault.sol
- AssetVault.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most resent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Umami GLP Vaults smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. Part of this work includes writing a test suite using the Hardhat testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.	03	Testing contracts logic against common and uncommon attack vectors.
02	Cross-comparison with other, similar smart contracts by industry leaders.	04	Thorough manual review of the codebase line by line.

Executive Summary

There were no critical issues found during the audit, alongside 46 issues with different levels of severity. They are described in detail in the “Complete Analysis” section.

Contracts are well written and structured. Some findings during the audit shall have an impact on contract performance and security, so it is not fully production-ready, but minor editions by the development team shall make it ready for production.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Umami team and the Umami team is aware of it, but they have chosen to not solved it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Cast issue for negative value	High	Resolved
2	First Depositor Issue	High	Resolved
3	Pricing equation is not functioning properly	High	Resolved
4	Function fail silently while in similar context it reverts	High	Resolved
5	Using solmate safeTransfer OR safeTransferFrom without code check	Medium	Resolved
6	ChainLink successfull response not implemented correctly	Medium	Resolved
7	Unexpected tokens with fees will break the vault	Medium	Resolved
8	Logic error in whitelist validation	Medium	Resolved
9	Unchecked possible zero address	Medium	Resolved
10	Unchecked Chainlink sequencer uptime	Medium	Resolved
11	Unchecked transfer result	Medium	Resolved
12	Unreachable code	Medium	Resolved
13	Decimals mismatched	Medium	Resolved
14	Method isTimeout signals a false positive	Medium	Resolved
15	Unsafe casting possibility	Medium	Resolved

#	Title	Risk	Status
16	sumColumns not properly looped	Medium	Resolved
17	Unexpected length of array returned	Medium	Resolved
18	Unchecked possible zero address	Low	Resolved
19	Array does not have a fixed length	Low	Resolved
20	Variable not used	Low	Resolved
21	Parameter not used	Low	Resolved
22	Pragma version lock	Low	Resolved
23	Config setters not emitting events	Low	Resolved
24	Unresolved TODO may result in revert	Low	Resolved
25	Approving more amount than the actual transfer	Low	Resolved
26	Redundancy in data might cause conflict	Low	Acknowledged
27	Unsafe casting of array elements	Low	Resolved
28	Remove unnecessary imports	Informational	Resolved
29	Emission of events omitted	Informational	Resolved
30	Pre-conditions not added	Informational	Resolved
31	No reason to override	Informational	Resolved
32	Missing non zero address check	Informational	Resolved
33	Missing licenses	Informational	Acknowledged
34	Duplicate code for contract check	Informational	Resolved
35	Redundant not null check	Informational	Resolved

#	Title	Risk	Status
36	Redundant comparison for uint param	Informational	Resolved
37	Missing function body	Informational	Resolved
38	Refactor function	Informational	Resolved
39	Refactor revert statements	Informational	Resolved
40	Refactor function	Informational	Resolved
41	Unused constant BIPS	Informational	Resolved
42	Redundant implementation of method	Informational	Resolved
43	Iterative multiplication causing unnecessary computation	Informational	Resolved
44	Solidity version in two contracts is not compatible with project	Informational	Resolved
45	Methode does not have an implementation in the body	Informational	Resolved
46	Contract have two pragma declarations	Informational	Resolved

HIGH | RESOLVED

Cast issue for negative value

In contract VaultFeeManager, function _getVaultRebalanceFees, the userPositionDelta is cast to uint256. If the userPositionDelta is negative, it will be cast to a very large positive number, and this could lead to unexpected behavior. The userPositionDelta should be cast to uint256 only if it is positive, otherwise the value of ' -userPositionDelta should be cast to uint256 .

Recommendation:

If the userPositionDelta is negative, it should be cast to uint256 as follows:

```
uint256 lockedBalanceSansUserDelta = userPositionDelta > 0
  ? currentBalance - uint256(userPositionDelta)
  : currentBalance + uint256(-userPositionDelta);
```

HIGH | RESOLVED

First Depositor Issue

Contract AssetVault is implementing the standard ERC4626, thanks to their design these types of vaults are subject to a share price manipulation attack that allows an attacker to steal underlying tokens from other depositors (this is a known issue of Solmate's ERC4626 implementation)
POC: https://hackmd.io/@T-egO1mKQkWZYwdNtLdSsw/Bk5rkHGRi (example was simplified to be easier to understand)

Recommendation:

In the deposit function of AssetVault, consider requiring a reasonably high minimal amount of assets during first deposit. The amount needs to be high enough to mint many shares to reduce the rounding error and low enough to be affordable to users.

Note:

Umami DAO will seed the vault first and run for days before opening the vault to the whitelist depositors. This will ensure the first depositor vulnerability can be used and allow the vault to run smoothly before user funds are added

HIGH | RESOLVED

Pricing equation is not functioning properly

GlpPricing.sol - In method usdToGlp, the result is divided by 1e24 to adjust units, but the placement of 1e24 in this context ends up having a result of zero for reasonably given usdAmount. Current formula is:
return ((usdAmount) / (glpPrice(maximise))) / 1e24;

Recommendation:
In order to match usdToGlp with glpToUsd , we estimate that the equation should be:
return ((usdAmount) * 1e24 / (glpPrice(maximise)));

HIGH | RESOLVED

Function fail silently while in similar context it reverts

SwapLibrary.sol - in _swapTokenExactOutput , the method returns `0` if token balance is zero. While in other context if balance is not enough to fulfill the swap it will revert. In that regard, this method has the potential to fail silently whilst the transaction succeeds, but the output state does not meet the expectation.
What makes this method even more critically severe is the usage of it in swapTokensExactOutput() as it does not validate the return value of method _swapTokenExactOutput. This might lead to loss of funds if the user of the library takes fund transfers decisions based on the success of the method call. Similar comment on _swapTokenExactInput.

Recommendation:
According to the usecase , devs can force method to revert or check the return of the private methods and act accordingly.

MEDIUM | RESOLVED

Using solmate safeTransfer OR safeTransferFrom without code check

All the contracts are making heavy use of solmate's library (e.g. function transferAsset from AggregateVault contract) the main usecase of this collection is to optimize the gas usage as much as possible, for that to be achieved there have been some compromised made by the library developers. This is a known issue while using solmate's libraries. Hence this may lead to miscalculation of funds and may lead to loss of funds, because if safetransfer() and safetransferfrom() are called on a token address that doesn't have a contract in it, it will always return success, bypassing the return value check. Due to this, the protocol will think that funds have been transferred successfully, and records will be accordingly calculated, but in reality, funds were never transferred. So this will lead to miscalculation and possibly loss of funds

Recommendation:
Use openzeppelin's safeERC20 or implement a code size check

Note:
All tokens that will be transferred using solmates transfer libs will only be well-defined tokens trading in GLP.

MEDIUM | RESOLVED

ChainLink successfull response not implemented correctly

The ChainLinkWrapper contract is not properly implementing the case where the oracle is responding successfully, this is originating from function *_getCurrentChainlinkResponse L#168*, the function is not assigning any value to the field success field from structure *OracleResponse*, this will make the successful call to fail in the check from function *_isBadOracleResponse*

Recommendation:

Assign the true value for the sucess field from the response struct
More explanations → Untitled document

Note:

Chainlink wrapper was removed, unused
https://github.com/UmamiDAO/V2-Vaults commit/4419ae6e85f59ef5c3d711ec0c8f7b942620fe90

MEDIUM | RESOLVED

Unexpected tokens with fees will break the vault

Contract AssetVault is a type of vault that supports only one ERC20 tokens, so for each token there will be a different vault deployed, based on the initial premise there will be quite a lot of tokens supports (e.g WBTC, WETH, USDC, LINK, UNI, etc...) .
And the list may extend in the future. However, if any of these tokens will start charging fee on transfers (on other blockchains USDT already charge fees on transfers, on Ethereum the logic is already there, but the fee is set as 0), or if you will ever extend the list of vaults with tokens that have fees, the logic will be broken

Recommendation:
To avoid this situation, use a common pattern, where you check the balance between operations.

Note:
We will remain with the assumption that none of these tokens, nor any GLP tokens potentially added in the future, will have fees on transfer.

MEDIUM | RESOLVED

Logic error in whitelist validation

In contract AggregateVault, the peripherals are set one by one using the setPeriheral function, one of those peripherals is the Whitelist contract that it is used in the onlyWhitelisted modifier, because the contracts needs to be deployed multiple times and each time it needs to be configured there are big chances of mistakes in repetitive tasks, if the Whitelist contract will not be set, then it's value will be address(0) and the logic will not enter the second if to break revert the execution, it will simply con tinue normally.

```
590     modifier onlyWhitelisted(address _account) {        0xdapper
591         Whitelist whitelist = _getWhitelist();
592         if (address(whitelist) != address(0)) {
593             if (!whitelist.isWhitelisted(_account)) {
594                 revert("AggregateVault: not whitelisted");
595             }
596         }
597         _;
598     }
599
```

Recommendation:

Revert the execution if the whitelist contract is the default address.

Note:

Address(0) set for whitelist contract is assumed to have no whitelist option enabled so the skip is intentional if 0'd

MEDIUM | RESOLVED

Unchecked possible zero address

In contract UmamiPriceFeed.sol there are multiple cases where input addresses are not checked to not be null/zero. For example, the setGov function sets the gov address, so in case the input address is by accident send as zero address it can lock the contract as the setter has the 'onlyGov' modifier. This also occurs in functions setGmxVaultPriceFeed and setSecondaryPriceFeed.

Recommendation:
Add a sanity check for the input addresses to not be null/zero

MEDIUM | RESOLVED

Unchecked Chainlink sequencer uptime

In contract ChainlinkWrapper there are no checks for the uptime status of the Chainlink L2 sequences. It's recommended that L2 oracles fetch the uptime for the Chainlink sequencer to ensure the data returned can be trusted. This can lead to errors on the L2 chain as the price data is not updated correctly. For a more detailed explanation, check the chainlink docs and code snippet: https://docs.chain.link/data-feeds/l2-sequencer-feeds.

Recommendation:
check if the sequencer is active and act on the response accordingly

Note:
Chainlink wrapper was removed, unused
https://github.com/UmamiDAO/V2-Vaults/commit/4419ae6e85f59ef5c3d711ec0c8f7b942620fe90

MEDIUM | RESOLVED

Unchecked transfer result

In contract AssetVault in function claimWithdrawalRequest at line 126 the result of the 'asset.transfer' is not checked. The value returned by the transfer function from ERC20 standard should be checked, as implementations differ.

Recommendation:
use safeTransfer

MEDIUM | RESOLVED

Unreachable code

Contract GmxPositionManager, function 'positionMargin', has two if conditional statements one after the other that check the same condition. In the first if statement however, if the condition is true, the function will simply return. This will make the code in the second if statement to never be reachable. The intention might be to just return the default values, in this case '0' and 'true' and the fix in this case is to remove line 342 entirely.

```
342     if (position.key == bytes32(0)) return (_margin, true);
343     if (position.key == bytes32(0)) {
344         _margin = 0;
345         _isLong = true;
346     } else {
```

Recommendation:
Re-write the if statements in a way that there's no unreachable branch (removing the first 'if' seems most natural).

MEDIUM | RESOLVED

Decimals mismatched

VaultLifecycle.sol - in method `rollover` we have this line:

```
ShareMath.sharesToAsset(params.currentQueuedWithdrawShares, newPricePerShare, params.decimals);
```

It is noted in the contract that `params.decimals` refers to the decimals of the asset ERC20. Given that the equation of `sharesToAsset` is:

```
(shares * assetPerShare) / 10**decimals
```

These equations show that the result is in decimals of shares ERC20 while it should be in decimals of asset ERC20.

Example:

- Given asset: usdc with decimals = 6
- decimals of shares ERC20 = 18
- `assetPerShare` = 2 usdc/share which is `2*10**6`
- `shares` = 10 share which is `10*10**18`
- result = `10 * 10**18 * 2 * 10**6 / 10**6 = 20 * 10**18`

The result given is too much for the usdc decimals, it was expected to be 20 * 10**6 instead.
As for pricePerShare, we have a similar situation given by this:

```
(singleShare * totalBalance) / totalSupply
```

as `singleShare = 10**decimals, totalBalance` points at asset and `totalSupply` points at shares. We end up having a formula that does not match the expected result. For method `pricePerShare` though, the only way to solve this is to assert that asset ERC20 decimals is the same as shares ERC20 decimals. Issue is existing in `ShareMath.sol`, need to do the following recommendation.

Recommendation:
Fix it in VaultLifecycle doc that `params.decimals` is referring to shares decimals. Also for `ShareMath.sol`, in `pricePerShare` adjust the formula to put in consideration decimals of both asset and shares. Like:

```
return totalSupply > 0? (singleShare * totalBalance) /totalSupply : singleShare* assetUnit/sharesUnit
```

where `assetUnit = 10**assetDecimals` and `sharesUnit = 10**sharesDecimals`

MEDIUM | RESOLVED

Method isTimeout signals a false positive

TimeoutChecker.sol - Method `isTimeout` returns true in a specific case (block.timestamp < timestamp). This case contradicts the fact that method signals `true` when `block.timestamp > timestamp` by a margin of `timeout`. This makes it susceptible to false signaling.

Recommendation:
Better revert if `block.timestamp < timestamp` to avoid underflow and false signaling.

MEDIUM | RESOLVED

Unsafe casting possibility

NettingMath.sol - in `vaultExposureInt` - casting uint to int is unsafe without validation. The result should be kept positive, as it is originally a multiplication of 2 uint256s `(glpHeld * glpComposition[i])`.

Recommendation:
Suggest the use of a SafeCast library, like this one used here by Quoter on arbiscan `0xb27308f9F90D607463bb33eA1BeBb41C27CE5AB6`

MEDIUM | RESOLVED

sumColumns not properly looped

Solarray.sol - In `sumColumns(int256[5][4] memory _array)`, the first for-loop which is supposed to loop on the first dimension of `_array` through `i`, is including 4 iterations (i.e. since `_array.length` returns 4) but it should be going through 5 iterations. The issue arises from assuming that both dimensions of the array are equal and represented as `_array.length`.

Recommendation:
first loop iterations should be 5 not 4. First for-loop limit be replaced by `_array[0].length`. And return length of array is int256[5] not int256[4].

MEDIUM | RESOLVED

Unexpected length of array returned

Solarray.sol - In `scaleArray - function scaleArray(uint256[4] memory _array, uint256 _scale) internal pure returns (uint256[5] memory _retArray) {`

`_retArray` is of length 5 while it should be 4.

Recommendation:
Change return to `(uint256[4] memory _retArray)`.

LOW | RESOLVED

Unchecked possible zero address

In contract Auth.sol, inside the constructor the _auth parameter can be zero and is not checked. In case the address is zero, the contract would not function properly and there is no fallback method to set the correct implementation after deployment.

Recommendation:
Add a sanity check for the to address variable to not be zero and revert otherwise.

LOW | RESOLVED

Array does not have a fixed length

In contract AggregateVaultStorage, struct AvStorage at line 77 there is an array called swapFee, that array will be setted as the same time as the vaults, in function setAssetVaults from AggregateVault contract, and in that function the length of the arrays is checked to be equal, at this moment the assetVaultEntiries have a fixed length of 5 items, however swapFee is not declared from the beginning as a fixed array of 5 items.

Recommendation:
Declare swapFee as a fixed array length of 5 items.

LOW | RESOLVED

Variable not used

In contract AggregateVault, at line 62, the variable WETH is hardcoded and declared immutable, however, it is not used anywhere in the contract.

Recommendation:
Remove the variable.

LOW | RESOLVED

Parameter not used

In contract AssetVault, at line 117, function requestRebalanceWithdraw, parameter owner it is never used.

Recommendation:
Remove the owner parameter.

LOW | RESOLVED

Pragma version lock

It's recommended to have the same compiler and flags that the contracts were tested with the most. This way, it reduces the risk of introducing unknown bugs by accidentally deploying with the latest version that might not be stable yet.

Recommendation:
Lock pragma versions

LOW | RESOLVED

Config setters not emitting events

In contract UmamiPriceFeed.sol in functions setGov, setGmxVaultPriceFeed, setMaxStrictPriceDeviation and setSecondaryPriceFeed there are no events emitted. It's recommended to emit events when config parameters change, as these can server as a history of changes that occured and also makes the changes observable offchain for subscribed listeners.

Recommendation:
Emit events for the corresponding functions

LOW | RESOLVED

Unresolved TODO may result in revert

Contract GmxPositionManager, function 'positionMargin', has an unresolved issue marked with a TODO that may result in an integer underflow, in case the 'collateral' is lower than 'amount'.

```
301         if (isProfit) {
302             return collateral + amount;
303         } else {
304             // TODO: assuming position is not underwater
305             return collateral - amount;
306         }
```

Recommendation:
Fix TODO by assuming position may be underwater.

LOW | RESOLVED

Approving more amount than the actual transfer

SwapLibrary.sol - in `_swapTokenExactOutput` we have:

```
_token.safeApprove(address(_swapRouter), _assetAmountIn);
```

Where `_assetAmountIn` refers to the maximum input amount in the swap operation that is required for slippage. Approving an amount more than actually needed is unsafe, therefore, extra care might be needed by the user of the library to remove the extra allowance that is approved here.

Recommendation:
Two ways can help:

- Prestimate the amountIn needed exactly by using `Quoter` contract in arbitrum `0xb27308f9F90D607463bb33eA1BeBb41C27CE5AB6`. Here is a code sample on how this can be used:

```
uint256 estimatedAmountIn = _quoter.quoteExactOutput(
    path,
    amountOutExact
);
_token.safeApprove(address(_swapRouter), _estimatedAmountIn)
```

- Override the approval by zero after the swap:

```
_token.safeApprove(address(_swapRouter), 0);
```

LOW | ACKNOWLEDGED

Redundancy in data might cause conflict

SwapLibrary.sol - in all methods of the library we have `swapData` and `fromAsset`. `fromAsset` is actually part of `swapData's` path variable. Passing conflicting data (i.e. different `fromAsset` and `path`) shall lead to undesireable results.

Recommendation:
Extract `fromAsset` from the path using byte decoding.

LOW | RESOLVED

Unsafe casting of array elements

Solarray.sol - In `intToUintArray`, elements of this array are being cast to uint without confirming they are greater than or equal to zero (i.e. within the domain of uint256). Also, similar issue in same contract in method `arrayDifference` where this `int256(_base[i])` considered unsafe casting without a prior validation.

Recommendation:
Suggest the usage of a SafeCast library.

INFORMATIONAL | RESOLVED

Remove unnecessary imports

In many contracts, the console is imported from “forge-std/console.sol”. Since the console module is primarily used for testing and debugging, it is not necessary or desirable to include it in a production contract. Removing this statement will help keep the codebase clean and efficient, while removing any unnecessary dependencies.

Recommendation:
Remove unnecessary import for console.

INFORMATIONAL | RESOLVED

Emission of events omitted

In contract VaultFeeManager, the functions handleDeposit, handleWithdraw, handleRebalanceRequest, handleGlpRewards are not emitting any events. Adding events is a good practice that enhances transparency and traceability of the smart contract operations. The event should capture all relevant information such as fees generated and relevant parameters.

Recommendation:
Add events for all of these functions to increase the transparency of the contract.

INFORMATIONAL | RESOLVED

Pre-conditions not added

In contract AggregateVault, functions openRebalancePeriod and closeRebalancePeriod have no preconditions added and todo comments left to add them.

Recommendation:
Remove the todo comments and add the pre-conditions.

INFORMATIONAL | RESOLVED

No reason to override

In contract AssetVault the hooks beforeWithdraw and afterWithdraw are overridden in the contract, however there is no reason to override the functions as there is no changes in them.

Recommendation:
Remove the unnecessary override.

INFORMATIONAL | RESOLVED

Missing non zero address check

In contract GmxPositionManager inside the constructor, the input parameters are not checked to be non zero. This can lead to misconfiguration and cause issues later while executing the contract.

Recommendation:
Check input addresses to be non zero

INFORMATIONAL | ACKNOWLEDGED

Missing licenses

Some of the contracts in the project are missing the license identifier, for example, GlpHandler.sol. Most of the contracts that have the license specified use the UNLICENSED identifier, so it's unclear if the ones without a version are intended to have No License (None). Even if that's the case, it's recommended to specify the license.

Recommendation:
Specify licenses for all contracts

INFORMATIONAL | RESOLVED

Duplicate code for contract check

In contract BaseWrapper.sol the functions isNullableOrContract and isContract perform the inline assembly extcodesize check for an address and assert whether the. This check is duplicated in both functions. Also note that extcodesize returns `0` if it is called from the constructor of a contract.

Recommendation:
Extract the code in a separate function or use Address.isContract from Openzeppelin

INFORMATIONAL | RESOLVED

Redundant not null check

In contract ChainlinkWrapper.sol in function setFlagContract at line 52 there's a check for the input address to not be null. This check is redundant as the function already uses the modifier 'notNull' which performs the same check.

Recommendation:
Remove line 52

INFORMATIONAL | RESOLVED

Redundant comparison for uint param

In contract ChainlinkWrapper.sol in function _isBadOracleResponse at line 255 the check verifies whether the _response.answer value is smaller or equal to `0`. Given the parameter is of type uint it can never be true for the smaller than case.

Recommendation:
Replace smaller than or equal operator (<=) with equal operator (==)

INFORMATIONAL | RESOLVED

Missing function body

Contract GlpHandler has a function, 'mintGlpUsdgAmount', that has nobody.

Recommendation:
Implement the function or make it private since it is never used externally.

INFORMATIONAL | RESOLVED

Refactor function

Contract GlpHandler, function 'previewGlpBurn', has a function signature that states that '_amtOut' will be returned, but in actuality, another value will be.

Recommendation:
Remove '_amtOut' from the function signature and declare it inside the function body.

INFORMATIONAL | RESOLVED

Refactor revert statements

Contract ChainlinkWrapper contains constructions like shown below that can be replaced with 'require' statements.

```
67        if (_isBadOracleResponse(currentResponse)) {
68            revert ResponseFromOracleIsInvalid(_token, _priceAggregator);
69        }
```

Recommendation:
Introduce require statements instead of imbricated revert and if statements.

INFORMATIONAL | RESOLVED

Refactor function

Contract GmxPositionManager, function 'gmxPositionCallback', has a function that contains an if that checks a condition, has a huge body, and on the else body, it has only a revert instruction.

Recommendation:
Refactor the if to check the '!_isExecuted' and revert to the 'true' branch and keep the rest of the body without the extra indentation.

INFORMATIONAL | RESOLVED

Unused constant BIPS

VaultMath.sol - unused constant `BIPS` declared in the library.

Recommendation:
Better be removed if not needed.

INFORMATIONAL | RESOLVED

Redundant implementation of method

VaultMath.sol - methods `getSlippageAdjustedAmount` and `checkNettingSlippage` are the same implementations.

Recommendation:
Use only one and remove the other.

INFORMATIONAL | RESOLVED

Iterative multiplication causing unnecessary computation

VaultMath.sol - method `arraySum(int256[5] memory _array)` is computationally inefficient since each element of `_array` is multiplied by 1e18.

Recommendation:
Make the summation without multiplication and multiply `sum` once outside the for loop after getting the summation done.

```
for (uint256 i = 0; i < _array.length; i++) {
    sum += _array[i] ;
}
sum *= 1e18;
```

INFORMATIONAL | RESOLVED

Solidity version in two contracts is not compatible with project

TimeoutChecker.sol & VaultStorage.sol - Those two contracts' solidity version compilation is locked on `0.8.7` while almost the rest of the project compiles with `^0.8.17`. The versions cannot match up since.

Recommendation:
This is solved consequently if the project solidity versions of contracts are all fixed on one version.

INFORMATIONAL | RESOLVED

Methode does not have an implementation in the body

PositionMath.sol -

```
function pnlFromPriceChange(
    uint256 oldPrice,
    uint256 newPrice,
    uint256 positionSize
) external pure returns (uint256) {
   // todo
}
```

Recommendation:
Remove if unneeded.

INFORMATIONAL | RESOLVED

Contract have two pragma declarations

In contract UmamiPriceFeed, there are two pragma declarations at the beginning of the file, this fact will make the project impossible to compile in a hardhat environment

Recommendation:
Remove the unnecessary pragma declaration.

	BaseHandler.sol GlpHandler.sol BasePositionManager.sol GmxPositionManager.sol PositionManagerRouter.sol BaseSwapManager.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

	GmxSwapManager.sol OneInchSwapManager.sol GlpPricing.sol NettingMath.sol ShareMath.sol SwapLibrary.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

	VaultLifecycle.sol VaultStorage.sol Multicall.sol PositionMath.sol Solarray.sol TimeoutChecker.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

	VaultMath.sol BaseWrapper.sol ChainlinkWrapper.sol UmamiPriceFeed.sol GlpRebalanceRouter.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

	NettedPositionTracker.sol VaultFeeManager.sol AggregateVaultStorage.sol AggregateVault.sol AssetVault.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

As a part of our work assisting Umami GLP Vaults in verifying the correctness of their contracts code, our team was responsible for writing integration tests using the Hardhat testing framework.
The tests were based on the functionality of the code, as well as a review of the Umami GLP Vaults contracts requirements for details about issuance amounts and how the system handles these.

Contract: ChainlinkWrapper

setFlagSEQ()
✓ Should setFlagSEQ (85ms)
setFlagContract()
✓ Should setFlagContract (55ms)
addOracle()
✓ should add oracle (1188ms)
✓ should remove oracle (1175ms)
retriveSavedResponses()
✓ should retrive saved responses (54ms)
getCurrentPrice()
✓ should get current price
getLastPrice()
✓ Should get last price
getExternalPrice()
✓ Should get external price (50ms)
isBadOracleResponse()
✓ should return correct value isBadOracleResponse() (77ms)
✓ should return correct value isBadOracleResponse() (140ms)

contract: GmxPositionManager

configMemory()
✓ Should configure configMemory
setMarginFeeBasisPoints()
✓ should set margin FeeBasis Points (79ms)
setToleranceBps()
✓ Should set ToleranceBps (49ms)
setReferralCode()
✓ should not set referral code (57ms)

setReferralCode()
✓ should set referral code successfully
setPreferredStablecoinToken()
✓ Should set preferred stablecoinToken (130ms)
increasePosition()
✓ Should increase position (214ms)
✓ Should not increasePosition (157ms)
✓ Should revert when increasePosition is called with balance less than exection fee (160ms)
✓ Should revert with InvalidParams, NotLongableToken (138ms)
✓ Should revert with NotLongableToken (1710ms)
✓ should revert if balance is less than executionfee (197ms)
✓ should successfully increase position with isLong false (210ms)
✓ Should revert with InsufficientAvailableOI (185ms)
✓ Should revert with InsufficientLiquidity (206ms)
✓ Should revert with NotShortableToken (126ms)
decreasePosition()
✓ It should successfully decrease position (375ms)
✓ Should revert insufficient balance (352ms)
increaseMargin()
✓ It should increase margin (441ms)
decreaseMargin()
✓ Should decrease margin (386ms)
preferredStablecoinToken()
✓ It should set preferred stablecoin token (91ms)
preferredStablecoinTokenDecimals()
✓ preferredStablecoinTokenDecimals (93ms)
getPosition()
✓ Should get position (1739ms)
updatePosition()
✓ Shpould update position (1770ms)
getPositionPnl()
✓ Should get position Pnl (322ms)
getPositionFees()
✓ Should get position fees (311ms)
positionMargin()
✓ should get position margin (1970ms)
positionMargin()
✓ Should get position margin by index token (1992ms)
positionNotional()
✓ Should get position notional (357ms)
✓ Should get position notional key not zero bytes (1711ms)

positionNotional 2

✓ positionNotional 2 (1687ms)

✓ Should get position notional by index token (1724ms)

gmxPositionCallback()

✓ should call gmxPositionCallback with islong true (1743ms)

✓ Should call gmxPositionCallback with isLong as false (1733ms)

✓ should gmxPositionCallback with isLong value as false and sizeDelta equal to 0 (1776ms)

syncCachedPosition()

✓ Should sync CachedPosition (1732ms)

updateCachedPosition()

✓ Should update Cached Position (1691ms)

updateCachedPosition()

✓ Should update cached position (1709ms)

gmxPositionCallback()

✓ Should revert with OnlyPositionRouter, UnknownAccount (175ms)

✓ gmxPositionCallback isIncrease should revert UnknownAccount (115ms)

✓ gmxPositionCallback isIncrease is successfully (139ms)

callbackSigs()

✓ should call callbackSigs

contract: PositionManagerRouterMock

updateHandlerContract()

✓ should update HandlerContract

✓ should updateHandlerContract callbackSigs (57ms)

updateSwapHandler()

✓ Should successfully update swapHandler (38ms)

updateDefaultHandlerContract()

✓ Should update defaultHandlerContract

execute()

✓ Should execute successfully

✓ should revert with UnknownHandlerContract

executeWithCallbackHandler()

✓ should execute With Callback Handler (112ms)

✓ should revert UnknownHandlerContract

executeSwap()

✓ should execute swap (45ms)

✓ should execute swap (91ms)

✓ should revert UnknownHandlerContract

tests fallback function

✓ Should call fallback function (115ms)

✓ revert with CallbackHandlerNotSet (88ms)

✓ should revert with UnknownCallback (58ms)

✓ should revert with UnknownCallback

✓ should revert handleCallback (412ms)
✓ tests delegateCall (594ms)
✓ tests OnlySelf (406ms)

Contract: UmamiPriceFeed

setGov()
✓ should set set gov address (52ms)
setMaxStrictPriceDeviation()
✓ should set max strict price deviation (50ms)
setUseV2Pricing()
✓ should set UseV2Pricing (39ms)
setIsAmmEnabled()
✓ should set isAmmEnabled (77ms)
setIsSecondaryPriceEnabled()
✓ should set is secondaryPriceEnabled (50ms)
setSecondaryPriceFeed()
✓ should set secondaryPriceFeed (49ms)

AggregateVault

✓ Should be able to set asset vaults (204ms)
✓ Should be able to handle deposit (572ms)
✓ Should be able to handle withdrawals (669ms)
✓ Should be able to handle rebalance request (528ms)
✓ Should be able to open rebalance period (292ms)
✓ Should be able to close rebalance period (1485ms)
✓ Should be able to add position manager (93ms)
✓ Should be able to set checkNetting status (79ms)
✓ Should be able to check netting constraint (177ms)
✓ Should be able to set vault fees (92ms)
✓ Should be able to set netted threshold (83ms)
✓ Should be able to update fee watermark balance (183ms)
✓ Should be able to get vault pps (119ms)
✓ Should be able to previewWithdrawalFee (118ms)
✓ Should be able to preview deposit fee (110ms)
✓ Should be able to set fee watermarks (94ms)
✓ Should be able to pause deposits (257ms)
✓ Should be able to unpause deposits (268ms)
✓ Should be able to set peripheral (446ms)
✓ Should allow multicalls (128ms)
✓ Should allow cycles to be made (106ms)
✓ Should call onlyconfigurator (90ms)
✓ Should call mockVaildateExecuteCallAuth (89ms)

AggregateVaultHelper

- ✓ Should be able to remove position manager at (153ms)
- ✓ Should be get vault pps (575ms)
- ✓ Should be able to reduce glp (95ms)
- ✓ Should be able to roll to next epoch (81ms)
- ✓ Should be able to do cycles (833ms)
- ✓ Should be able to settle internal pnl (146ms)
- ✓ Should be able to set rebalance state (152ms)
- ✓ Should be able to set external positions adjusted (91ms)
- ✓ Should be able to update netting checkpoint price (170ms)
- ✓ Should be able to rebalance glp position (732ms)
- ✓ Should be able to handle glp rewards (370ms)
- ✓ Should be able to handle glp rewards if no rewards (268ms)
- ✓ Should be able to reset glp attributes slippage (327ms)
- ✓ Should be able to set vault glp attribution (88ms)
- ✓ Should be able to get vault glp attribution (88ms)
- ✓ Should be able to process withdrawal requests (276ms)
- ✓ Should be able to call back sigs
- ✓ Should be able to get vault tvl (277ms)
- ✓ Should be able to reset checkpoint prices (91ms)
- ✓ Should be able to get last netted price (91ms)
- ✓ Should be able to get active aggregate positions (87ms)
- ✓ should be set active aggregate positions (88ms)
- ✓ should be able to get swap fees (87ms)
- ✓ should be able to get netted position (124ms)
- ✓ should be to get active external positions (126ms)
- ✓ should be able to get last glp composition (91ms)
- ✓ Should be able to get all asset vaults hedge attribution (780ms)
- ✓ Should be able to get vault state (171ms)
- ✓ Should be able to get rebalance state (188ms)
- ✓ Should be able to get vaultToAssetVaultIndex (72ms)
- ✓ Should be able to set asset vaults (95ms)
- ✓ Should be able to set asset vaults (70ms)
- ✓ Should be able to update epoch (172ms)
- ✓ Should be able to get withdrawal requests (114ms)
- ✓ Should be able to get position manager (74ms)
- ✓ Should be able to delegate call (61ms)

AssetVault

- ✓ Should be able to deploy correctly
- ✓ Should be able to update aggregate vault (190ms)
- ✓ Should be able to able to mint (64ms)
- ✓ Should be able to able to deposit (144ms)

- ✓ Should be able to able to withdraw (460ms)
- ✓ Should be able to get max withdraw (328ms)
- ✓ Should be able to get max redeem (123ms)
- ✓ Should be able to get pps (57ms)
- ✓ Should be able to redeem (466ms)
- ✓ Should request rebalance withdraw (306ms)
- ✓ Should allow withdrawal request for claims (126ms)
- ✓ Should be able to process withdrawal requests (103ms)
- ✓ Should be able to able pause deposit withdraw (132ms)
- ✓ Should be able to able to unpause deposit withdraw (175ms)
- ✓ Should be able to get max deposit
- ✓ Should be able to get max mint
- ✓ Should be able to get total assets (190ms)
- ✓ Should be able to convert asset to shares (133ms)
- ✓ Should be able to convert shares to asset
- ✓ Should be able to preview withdraw
- ✓ Should be able to preview mint (128ms)

GlpHandler

- ✓ Should be able to be deploy correctly
- ✓ Should be able to mint glp token (160ms)
- ✓ Should be to get callback sigs
- ✓ Should be able to preview glp burn (183ms)
- ✓ Should be able to get token price (93ms)
- ✓ Should be able to get usd to token (128ms)
- ✓ Should be able to convert usd to glp (66ms)
- ✓ Should be able to get glp dynamic aum (93ms)
- ✓ Should be able to get glp static aum (412ms)
- ✓ Should be able to get glp price (91ms)
- ✓ Should be able to get glp price given max status (126ms)
- ✓ Should be able to mint glp usdg amount (110ms)
- ✓ Should be able to get underlying glp tokens (300ms)
- ✓ Should be able to get glp composition (834ms)

GlpRebalanceRouter

- ✓ Should be get directional sum (88ms)
- ✓ Should be able to get net glp rebalance (229ms)

GMX SWAP Manager

- ✓ Should be to get callback sigs
- ✓ Should be able to swap tokens (332ms)

nettedPositionTracker

- ✓ Should be deployed correctly (70ms)
- ✓ Should be able to update zerosum threshold (216ms)
- ✓ Should be able to scale pnl to glp (45ms)

✓ Should be able to check negative difference (258ms)
✓ Should be able to settle netting position pnl (275ms)
✓ Should be able to check zero sum (48ms)

One Inch SWAP Manager

✓ Should be able to be deploy correctly
✓ should allow for swaps (143ms)

VaultFeeManger

✓ Should be deployed correctly (43ms)
✓ Should be able to get rebalance fess (241ms)
✓ Should get deposit fee (115ms)
✓ Should get withdrawal fee (302ms)

Whitelist

✓ Should be deployed correctly (41ms)
✓ Should be able to check if account is whitelisted (158ms)
✓ Should be able to update whitelist enabled (214ms)
✓ Should be able to update whitelist (230ms)

190 passing (3m)

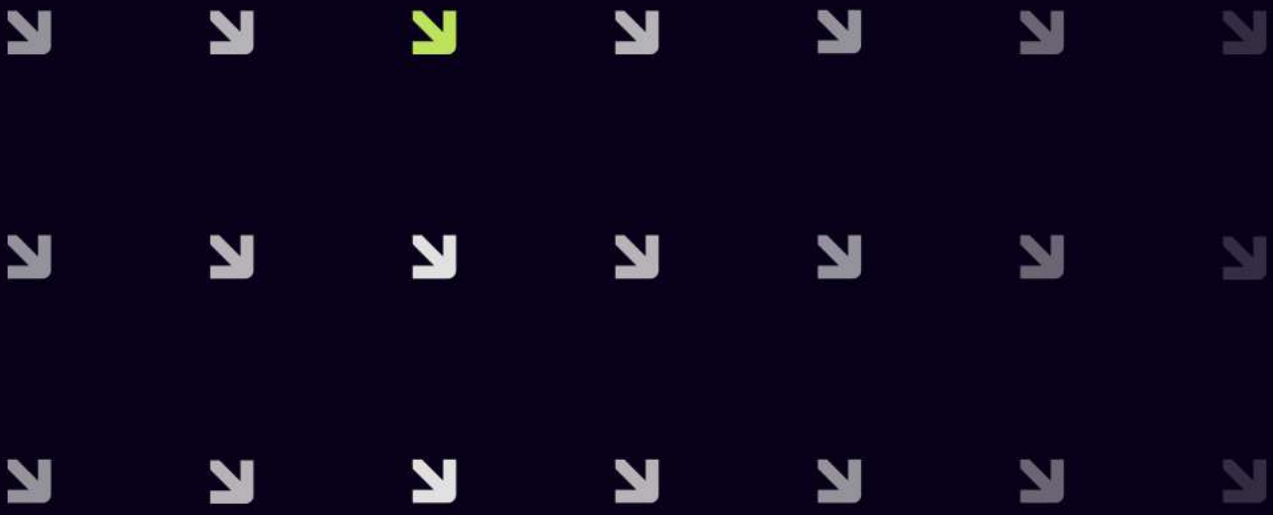
The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	% UNCOVERED LINES
PositionManagerRouter.sol	100	83.33	100	96.30	
ChainlinkWrapper.sol	93.75	80	100	87.36	
UmamiPriceFeed.sol	100	90.38	100	96.43	
GlpHandler.sol	100	100	100	100	
GmxPositionManager.sol	96.97	89.09	100	96.94	
GmxSwapManager.sol	100	100	100	100	
OneInchSwapManager.sol	100	100	100	75	
AggregateVaultHelper.sol	97.16	94.87	100	96.89	
GlpRebalanceRouter.sol	100	87.5	100	100	
NettedPositionTracker.sol	100	94.44	100	100	
VaultFeeManager.sol	100	100	100	100	
Whitelist.sol	100	100	100	100	
AggregateVaultStorage.sol	100	100	100	100	
AggregateVault.sol	100	95.45	100	100	
ssetVault	100	100	100	100	
All files	**98.34**	**92.08**	**100**	**97.29**	

We are grateful for the opportunity to work with the Umami team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Umami team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.



zokyo