UMAMI DAO

UMAMI GLP VAULTS
REPORT 2 OF 2

SMART CONTRACT AUDIT

zokyo

June 16th 2023 | v. 1.2

Security Audit Score

PASS

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

SCORE
95

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)

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 GLP Vaults 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

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

Last commit - 5d5623674575f5c35608e4da9b19b1f904ae6654

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
- AavePositionManager
- AaveIsolatedPositionAccount
- AaveUtils
- GmxAccountManager
- GmxPositionManagerStorage
- GmxPositionManagerUtils
- OdosSwapManger
- CorrelationRegistry

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's Security Team has followed best practices and industry-standard techniques to verify the implementation of Umami GLP Vaults smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. 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.

02 Cross-comparison with other, similar smart contracts by industry leaders.

03 Thorough manual review of the codebase line by line.

Executive Summary

No critical issues were identified during the audit, but two issues with high severity were discovered, as well as some with medium, low, and informational severity levels. These issues are comprehensively described in the "Complete Analysis" section. The contracts are well-written and well-structured.

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 Labs team and the Umami Labs 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	Price manipulation through Uniswap pool	High	Resolved
2	Unchecked shares can lead to stolen assets	High	Resolved
3	Unchecked _amountOut can lead to 100% slippage	Medium	Resolved
4	Possible call to zero address	Medium	Resolved
5	High hardcoded tolerance (slippage)	Medium	Resolved
6	Possible zero address for intermediary asset	Low	Acknowledged
7	AavePositionManager compatibility with Vaults might fail in some edge cases	Low	Acknowledged
8	Constant not used anywhere in code	Informational	Resolved
9	ap deposit invariant is not as expected	Informational	Resolved
10	Decrease position should not call _getOrCreateAccount	Informational	Resolved
11	Check for totalSupply equal 0 inside the withdraw function	Informational	Resolved
12	The fee for the flash loan is not checked	Informational	Resolved

HIGH | RESOLVED

Price manipulation through Uniswap pool

In contract UniswapV3SwapManager, the swap of tokens is done through the _swapTokenExactInput function, which first create the necessary parameters for the swap and then call the function exactInput from the Uniswap V3 Router. To swap tokens, uniswap is using liquidity pools, the pools are organized in different fee tiers, at the beginning there we're 3 tiers (0,05%, 0,3%, 1%), after a governance vote, they have also added the option for the 0.01% fee tier. The likelihood of adding new pools in the future and liquidity migrating to it or the change of non-existing pool is create an possible attack vector in this case. UniswapV3SwapManager is generating the swap path inside the _getSwapPath, it is also taking into consideration the existence of an intermediaryAsset for the easy of swap in case there are no available pairsLet's take the following example :

1. Let's say you want to swap LUSD with WETH and the intermediaryAsset is WETH, the usual fee for the WETH intermediayAsset is 0.05% because you noticed there is the most liquidity inthe majority of the pool, however in the case of LUSD-WETH pool, all of the liquidity is in the 0,3% pool, 1% and 0,05% pools liquidity is almost non-existent and the pool for 0,01% does not exist (which means it can be created and manipulated by an attacker), this will expose the protocol to a price manipulation attack that can result in either loss of funds or dos.
2. Let's say you want to swap USDT with USDC, intermediaryAsset is WETH, most of the liquidity in USDC-WETH is in the 0,05% pool, so is the case for USDT-WETH, however the pool for USDC-USDT pair with the biggest liquidity is the 0,01% fee, if you would go directly to the to the last pool, you will also benefit from a higher liquidity but you will also have a cheaper fee which will help you save funds, an clear example why pre-configuration of fee tires is not always a good decision.
3. Let's say you want to swap USDC-USDT, you initially configured for the 0,05% fee tier and everything is going fine, hoverwer a few months in the future the Uniswap dao governance vote to add another tier pool of 0,0% fees and all the liquidity will migrate to that pool, opening the protocol again to price manipulation attacks.

Recommendation:

To ensure the protocol is working properly no matter the conditions, drop the mechanism of fees configuration and dynamically look for the pool with the hight liquidity when creating the swap path.

HIGH | RESOLVED

Unchecked shares can lead to stolen assets

In contract AssetVaults, function `withdraw` takes as input parameters some assets, the receiver of the withdrawed assets and the shares owner. The first step is to add the withdrawal fees to the assets then convert the amount of assets + fees to shares after that check the allowances over that shares in case the msg.sender is different from owner, burn the shares and transfer the assets to the receiver and the fees to the fees recipient.This function have 3 low level weak points that an attacker can chain to withdraw assets without even owning any shares.
The weakpoints:

1. Allowance checking using a math formula and making the assumption it will revert thanks to the underflow.
2. Rounding down of the assets inside the assetToShares function
3. Not sanity checking the outputed value that represents the "shares"

Scenario :

1. Price per share is `1000e6` (1k usdc), usdc is 6 decimals and there are plantly of assets inside the vault
2. A malicious attacker that have never owned any shares or deposited any assets call the withdraw function with the following parameters assets = 8e2 - fees (for a simplified example), receiver = attacker address , owner = any address
3. The first step inside the logic is to first add the fees to the assets, as we we're saying in the above point, the assets = 8e2-fees so assets + fees = 8e2
4. Now the contract converts from assets to shares using the function "assetsToShares", and the math formula will be like this: 8e2 * 1e6 / `1000e6` = `0` (because of the rounding down), so variable 'shares' will have value `0`
5. Now the logic goes further inside the if condition because the msg.sedner != owner, it takes the allowances of the msg.sender over the owner and stores it in variable allowed, which will be `0`, now the logic is backfilling on the assumption that the execution will revert with an underflow because allowed - shares => underflow revert if allowed is `0`, however as shares is also `0`, it will be `0` - `0` = `0` which will not revert and continue the execution.
6. Now it will burn `0` shares from the owner, and as the ERC4626 version you are using is coming from the solmate library and is prioritizing gas consumption there is no check inside it that will prevent the burning of `0` assets.

7. The execution continue inside the aggregateVault and it will transfer the assets to the attack and the fees to the fee recipient.

This attack could be run inside a for loop in one transaction and use dark pools or flashbots and other MEV techniques to send multiple transactions and acaparte multiple blocks to create a bigger financial drain, also the Price Per Share is a very important factor here, the bigger it is (price per share) the easier it is to attack because you can use a bigger value for assets and still achieve a roun down that will lead to shares being `0`

Recommendation:

1. Add a sanity check to ensure shares can not be 0
2. Put the allowance check logic inside an internal function as it is used a lot around the code and it is just copy-pasted (redundant) and add a sanity check to revert if allowed = 0

MEDIUM | RESOLVED

Unchecked _amountOut can lead to 100% slippage

In contract AaveUtilsl, function `_tokenSwapOutAmount` will return in 0 when the token amount is a small value, resulting in an swap with a _minOut of 0 which on it's on with lead to a swap with 100% slippage which can be easily sandwich by a MEV bot for profit.

Recommendation:

Add a sanity check for the to ensure the output of `_tokenSwapOutAmount` and _minOut can never be 0.

MEDIUM | RESOLVED

Possible call to zero address

In the GmxAccountManager contract, when the _executeAccountSet function is used to make external calls to each account using assembly, the calls will still be executed even if some of the accounts have not been set properly (i.e., have a zero address). In this case, the execution will not be reverted.

Recommendation:

To ensure proper behavior, add a sanity check to ensure the target address is different from address(0).

MEDIUM | RESOLVED

High hardcoded tolerance (slippage)

In the AaavePositionManager contract the swap `TOLERANCE_BIPS` in bips is hardcoded to a high value (2%) relative to the usual slippage (0.5%) present in swap transactions. This can lead to smaller than expected out amounts given the in amount and it will make the contract an open target for MEV extractors that are using sandwich attacks.

Recommendation:

Allow for this value to be configurable at least at contract level in case it needs adjustments and start with an value of 0.5% for it.

LOW | ACKNOWLEDGED

Possible zero address for intermediary asset

In the AaavePositionManager contract the `intermediaryAsset` field inside the Config struct can be zero address. This leads will revert the call inside the uniswap contract, at function exactInputInternal #101r.

Recommendation:

Set this value while deploying the contract or make sure it’s non zero before making the uniswap call.

LOW | ACKNOWLEDGED

AavePositionManager compatibility with Vaults might fail in some edge cases

Aave V3 is configured to work with a few selected tokens, and the mixed of selected tokens is different on each chain, on the arbitrum chain the UNI tokens is not supported on Aave V3, the oracle will revert if you even try to fetch prices for it, however it is available on ethereum mainne version of Aave V3. If one of the Umami vaults will use UNI tokens as the underlying asset, that liquidity will not be compatible with AavePositionManager handler.

Recommendation:

Ensure a strategy where only selected vaults will work with compatible handlers to not let Keepers have surprises.

INFORMATIONAL | RESOLVED

Constant not used anywhere in code

In contract AaveUtils, constant INTEREST_RATE_MODE_STABLE is declared correctly as the stable more rate inside the Aave protocol is identified using the integer 1, however is never used inside Umami products.

Recommendation:

Remove the constant if you don't intend to use it.

INFORMATIONAL | RESOLVED

Cap deposit invariant is not as expected

In contract AssetVault, the deposit and mint functionalities have a hard cap to not allow the deposit of new assets after a certain amount, however the logic inside that condition (invariant) is not as you would expected, usually a cap needs to be hitted, here the cap can never be hitted because the sum of 'tvl + assets' always needs to be smaller then the cap for the deposit to success.

Recommendation:

Refactor the sign < (smaller) inside the condition to <= (smaller or equal) to allow the vault to hit it's cap.

INFORMATIONAL | RESOLVED

Decrease position should not call _getOrCreateAccount

In the AaavePositionManager contract the `_decreasePosition` function retrieves the Aave position account by calling `_getOrCreateAccount` function. The call to that function is redundant and unexpected in the given context, as it should call the `_getAccountOrRevert` function which reverts in case of an inexistent position instead of creating one.

Recommendation:

Change the call to `_getAccountOrRevert` as it makes more sense in the context.

INFORMATIONAL | RESOLVED

Check for totalSupply equal 0 inside the withdraw function

In the AssetVault contract, inside the "withdraw function, before converting the assets to shares, is checking if totalSupply = 0, this check is redundant because if totalSupply si equal with 0, there are no shares or assets to withdraw as the shares will be minted through the deposit and mint function and they will be burned throug the _burn function and the totalSupply variable is only incremented during the minting (of shares) and decremented over the burning of shares

Recommendation:

Remove the check totalSupply = 0 as it is not necessary.

INFORMATIONAL | RESOLVED

The fee for the flash loan is not checked

In the AaavePositionManager contract, specifically in the `receiveFlashLoan` function, it is important to ensure that the payment for the loan, which includes an additional fee on top of the borrowed amount, does not exceed a predetermined percentage of the loan amount. This fee, determined by a function parameter and based on an external protocol, needs to be checked to mitigate the risk of potential overcharging and protect the system from unexpected or excessive fees during flash loan transactions. Right now the balancer vault fee is 0, however that can change the in the future and you need to have a protocol that is antifragile.

Recommendation:

Define a constant or configurable parameter that represents the maximum allowable fee percentage. For example, you might set it to 1% or any other appropriate value. Verify that the fee amount does not exceed the predetermined percentage of the loan amount.

	BaseHandler.sol GlpHandler.sol BasePositionManager.sol GmxPositionManager.sol PositionManagerRouter.sol BaseSwapManager.sol GmxSwapManager.sol OneInchSwapManager.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

	GlpPricing.sol NettingMath.sol ShareMath.sol SwapLibrary.sol VaultLifecycle.sol VaultStorage.sol Multicall.sol PositionMath.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

	Solarray.sol TimeoutChecker.sol VaultMath.sol BaseWrapper.sol ChainlinkWrapper.sol UmamiPriceFeed.sol GlpRebalanceRouter.sol NettedPositionTracker.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

	VaultFeeManager.sol AggregateVaultStorage.sol AggregateVault.sol AssetVault.sol AavePositionManager AaveIsolatedPositionAccount AaveUtils GmxAccountManager
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

	GmxPositionManagerStorage GmxPositionManagerUtils OdosSwapManger CorrelationRegistry
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

We are grateful for the opportunity to work with the Umami GLP Vaults 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 GLP Vaults team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo