FORTITUDE

VAULTKA

Arbitrum Nitro Integration

SMART CONTRACT SECURITY AUDIT

FEBRUARY 2023

CONTENTS

EXECUTIVE SUMMARY

TYPES
Defi Yield Vault Auditing

METHODS
Automated Test, Manual Review, Static Analysis

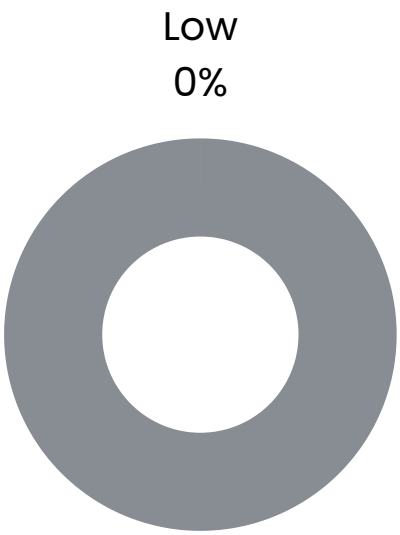
LANGUAGE
Solidity

COMMIT HASH
F15B53CE5D885CD598C4FC7364D94455BF7F8EDD

TIMELINE
Two days

REPOSITORY
HTTPS://GITHUB.COM/VAULTKA/VAULTKA-CONTRACTS

VULNERABILITY SUMMARY



1	0	1
Total Findings	Resolved	Unresolved

0 HIGH RISK

High risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.

0 MEDIUM RISK

Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform

0 LOW RISK

Minor risks do not compromise the overall integrity of the project, but they may be less efficient than other solutions.

1 INFORMATIONAL

Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code

SMART CONTRACT AUDIT SUMMARY

SMART CONTRACT AUDIT SUMMARY

The previous audit was conducted to assess the security and functionality of the target smart contract.

The audit findings identified two (2) known high vulnerabilities and nine (9) medium exposure.

Based on the audit findings, recommendations were made to improve the overall security and the functionality of the smart contract.

TARGET CONTRACT

https://github.com/RageTrade/delta-neutral-gmx-vaults/tree/8bea1afbe746387b1a66ea9357bd41fb1c74830

LANGUAGE
Solidity

TIMELINE
Oct 31 - Nov 14, 2022

PREVIOUSLY AUDITED BY
Sherlock

ABOUT VAULTKA

Sherlock, a leading blockchain security firm, audited the RageTrade/delta-neutral-gmx-vaults smart contract code, which is located at the repo:
8bea1afbe746387b1a66ea9357bd41fb1c74830b.

THE AUDIT INCLUDED THE FOLLOWING CONTRACTS:

- ERC4626/ERC4626Upgradeable.sol
- Libraries/DnGmxJuniorVaultManager.sol
- Libraries/FeeSplitStrategy.sol
- Libraries/SafeCast.sol
- Periphery/WithdrawPeriphery.sol
- Vaults/DnGmxBatchingManager.sol
- Vaults/DnGmxJuniorVault.sol
- Vaults/DnGmxSeniorVault.sol

In the report, some findings were identified and addressed by the development team. The report was thorough, and comprehensive and demonstrated a high level of expertise of Sherlock in smart contract security. Overall, the audit showed that the codebase was implemented securely and met industry best practices.

It is important to note that while no security system can be completely secure, the audit conducted by Sherlock provides a high level of assurance that the code is secure and that investors' funds are well protected.

BACKGROUND INFORMATION

Vaultka platform is a fork of Rage Trade that focused on building the most liquid, composable, and only omnichain ETH perpetual contract (powered by UNI v3) with additional use cases. The goal of Vaultka is to provide a platform that enables users to trade and manage their digital assets in a secure and efficient manner.

In order to achieve this goal, several smart contracts are within the scope of this report.

THE CONTRACT INCLUDES:

- ERC4626/ERC4626Upgradeable.sol,
- libraries/VodkaVaultManager.sol,
- libraries/FeeSplitStrategy.sol,
- libraries/SafeCast.sol,
- periphery/WithdrawPeriphery.sol,
- vaults/DnGmxBatchingManager.sol,
- vaults/VodkaVault.sol,
- vaults/WaterVault.sol.
- vaults/VaultProxyAdmin.sol

These contracts are responsible for managing the platform's operations and handling transactions between users. The ERC4626/ERC4626Upgradeable.sol contract is an upgradeable contract that allows for the platform to be updated as needed.

The libraries/VodkaVaultManager.sol contract manages the operations of the Vodka Vault.
The libraries/FeeSplitStrategy.sol contract handles the distribution of fees among platform participants.
The libraries/SafeCast.sol contract is used to safely cast values between different data types.
The periphery/WithdrawPeriphery.sol contract is responsible for handling withdrawals from the platform.
The vaults/DnGmxBatchingManager.sol contract manages the operations of the DnGmx Batching Manager.
The vaults/VodkaVault.sol contract manages the operations of the junior tranche which maintains a hedge for BTC and ETH basis the target weight on GMX.
The vaults/WaterVault.sol contract manages the operations of the senior tranche which acts as a lender of USDC for the junior tranche to borrow and hedge tokens using AAVE.

Overall, these contracts are critical to the functioning and security of the Vaultka as forked from the Rage Trade platform, and ensuring that they are secure is of the utmost importance.In order to achieve this goal, several smart contracts are within the scope of this report.

AUDIT SCOPE / VULNERABILITIES CHECKED

The list of vulnerability checks conducted on the contract internally includes but not limited to;

- Using block.timestamp.
- Using block.number.
- Reentrancy.
- Access controls
- Arithmetic Issues (integers Overflow/Underflow).
- Unchecked return value for low level call.
- Unsafe external calls
- Business logic contradicting the specification
- Short Address Attack
- Unknown Vulnerability.
- Centralization of power.
- Timestamp Dependence.
- Exception Disorder.
- Compiler version not fixed.
- Address hardcoded.
- Divide before multiply.
- Integer overflow/underflow.
- Dangerous strict equalities.
- Missing Zero Address Validation.
- Revert/require functions.

FINDINGS SUMMARY

After a thorough review of the contract code, architecture, and deployment, no critical security vulnerabilities were found, but some areas for improvement were identified. The code and architecture were found to be well-structured and implemented in accordance with some best practices for smart contract development. However, a few informational issues were discovered and documented for future consideration.

WE HAVE PERFORMED VARIOUS CHECKS, INCLUDING, BUT NOT LIMITED TO:

- Review of the contract logic and algorithms
- Analysis of the contract's potential attack surfaces
- Validation of the contract's security measures and protections
- Assessment of the contract's compliance with relevant standards and guidelines

Based on the audit findings, we have one recommendation to improve the overall functionality of the smart contract. While this recommendation does not directly relate to security risks, we believe it will be beneficial to the performance and user experience of the contract. We will continue to monitor the contract for potential security risks and address any new vulnerabilities that may arise in the future.

FINDINGS

ID	TITLE	SEVERITY	STATUS
FSA-01	Floating pragma	Informational	Unresolved

Unlock Pragma

Description

The contract makes use of the floating-point pragma >=0.8.0. Contracts should be deployed using the same compiler version and flags that were used during the testing process. The use of a floating pragma increases the risk of encountering compatibility issues with different versions of the compiler, as well as with future upgrades to the compiler. This can result in unintended behavior or vulnerabilities in the contract. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma.

```
pragma solidity >=0.8.0;
```

Recommendations

Avoid the use of a floating pragma is to specify a fixed pragma version that corresponds to the version of the compiler being used. This ensures that the contract is compiled using a known and tested version of the compiler, reducing the risk of compatibility issues and vulnerabilities. Additionally, it is good practice to regularly upgrade the fixed pragma version to take advantage of bug fixes and security upgrades in the compiler.

```
pragma solidity 0.8.0;
```

RECOMMENDATIONS

While we did not identify any critical security vulnerabilities during the audit, we did document some informational findings that may be of interest. Based on our review, we have no specific recommendations for improvement at this time. However, we remain committed to monitoring the contract for potential security risks and addressing any new vulnerabilities that may arise in the future.

CONCLUSION

In conclusion, the Vaultka smart contract is secure and free of any critical vulnerabilities. The contract's code and architecture have been implemented in accordance with best practices for smart contract development, and the deployment is secure and well-structured. We are confident in the security and stability of the contract and recommend it for use.

AUTOMATED TESTING AND VERIFICATION

The below-automated testing techniques are used to enhance coverage of certain areas of the token contract.

- Slither, a Solidity static analysis framework. Slither can statically verify algebraic relationships between Solidity variables. I used Slither to detect invalid or inconsistent usage of the contracts' APIs across the entire codebase.

While automated testing methods can enhance manual security evaluation, they cannot replace it completely. Each approach has its own limitations, for example, Slither may detect security features that do not hold up when translated into Solidity code.

Formalized and tested a variety of properties, from high-level ones to very specific and low-level ones in basic libraries like libraries/VodkaVaultManager.sol, libraries/FeeSplitStrategy.sol, and libraries/SafeCast.sol was performed by the team

Regarding property coverage, the core of the contract, consisting of the Withdraw Periphery, Deposit Periphery, DnGmxBatchingManager, VodkaVault, and WaterVault contract and its libraries, received substantial coverage.

The VodkaVault and WaterVault contracts contain the main business logic. It is the entry point for essential operations such as responsible for managing the platform's operations and handling transactions between users, along with other contracts.

AUTOMATED TESTS (SLITHER)

```
.....
..... HIGH WARNINGS .....

IERC20 is re-used:
    - IERC20 (node_modules/@aave/core-v3/contracts/dependencies/openzeppelin/contracts/IERC20.sol#7-80)
    - IERC20 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#9-82)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#name-reused

DnGmxJuniorVault.__gaps (contracts/vaults/DnGmxJuniorVault.sol#62) shadows:
    - ERC4626Upgradeable.__gaps (contracts/ERC4626/ERC4626Upgradeable.sol#32)
DnGmxSeniorVault.__gaps (contracts/vaults/DnGmxSeniorVault.sol#63) shadows:
    - ERC4626Upgradeable.__gaps (contracts/ERC4626/ERC4626Upgradeable.sol#32)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing

DnGmxJuniorVaultManager._executeOperationToken(DnGmxJuniorVaultManager.State,address,uint256,uint256,uint256,bool) (contracts/libraries/DnGmxJuniorVaultManager.sol#928-980) ignores return value by IERC20
(token).transfer(address(state.balancerVault),amountWithPremium) (contracts/libraries/DnGmxJuniorVaultManager.sol#954)
DnGmxJuniorVaultManager._executeOperationToken(DnGmxJuniorVaultManager.State,address,uint256,uint256,uint256,bool) (contracts/libraries/DnGmxJuniorVaultManager.sol#928-980) ignores return value by state.usdc.
transfer(address(state.balancerVault),usdcAmount + premium) (contracts/libraries/DnGmxJuniorVaultManager.sol#978)
SwapRouterMock.exactOutputSingle(ISwapRouter.ExactOutputSingleParams) (contracts/mocks/SwapRouterMock.sol#21-28) ignores return value by IERC20(params.tokenIn).transferFrom(msg.sender,address(this),params.
amountInMaximum) (contracts/mocks/SwapRouterMock.sol#25)
SwapRouterMock.exactOutputSingle(ISwapRouter.ExactOutputSingleParams) (contracts/mocks/SwapRouterMock.sol#21-28) ignores return value by IERC20(params.tokenOut).transfer(msg.sender,params.amountOut) (contracts/
mocks/SwapRouterMock.sol#26)
SwapRouterMock.exactInputSingle(ISwapRouter.ExactInputSingleParams) (contracts/mocks/SwapRouterMock.sol#30-34) ignores return value by IERC20(params.tokenIn).transferFrom(msg.sender,address(this),params.
amountIn) (contracts/mocks/SwapRouterMock.sol#31)
SwapRouterMock.exactInputSingle(ISwapRouter.ExactInputSingleParams) (contracts/mocks/SwapRouterMock.sol#30-34) ignores return value by IERC20(params.tokenOut).transfer(msg.sender,params.amountOutMinimum)
(contracts/mocks/SwapRouterMock.sol#32)
SwapRouterMock.exactOutput(ISwapRouter.ExactOutputParams) (contracts/mocks/SwapRouterMock.sol#36-53) ignores return value by IERC20(from).transferFrom(msg.sender,address(this),params.amountInMaximum)
(contracts/mocks/SwapRouterMock.sol#50)
SwapRouterMock.exactOutput(ISwapRouter.ExactOutputParams) (contracts/mocks/SwapRouterMock.sol#36-53) ignores return value by IERC20(to).transfer(msg.sender,params.amountOut) (contracts/mocks/SwapRouterMock.
sol#51)
SwapRouterMock.exactInput(ISwapRouter.ExactInputParams) (contracts/mocks/SwapRouterMock.sol#55-72) ignores return value by IERC20(from).transferFrom(msg.sender,address(this),params.amountIn) (contracts/mocks/
SwapRouterMock.sol#69)
SwapRouterMock.exactInput(ISwapRouter.ExactInputParams) (contracts/mocks/SwapRouterMock.sol#55-72) ignores return value by IERC20(to).transfer(msg.sender,params.amountOutMinimum) (contracts/mocks/
SwapRouterMock.sol#70)
DnGmxJuniorVault.withdrawFees() (contracts/vaults/DnGmxJuniorVault.sol#319-324) ignores return value by state.weth.transfer(state.feeRecipient,amount) (contracts/vaults/DnGmxJuniorVault.sol#322)
DnGmxSeniorVault.borrow(uint256) (contracts/vaults/DnGmxSeniorVault.sol#194-204) ignores return value by aUsdc.transfer(msg.sender,amount) (contracts/vaults/DnGmxSeniorVault.sol#203)
DnGmxSeniorVault.repay(uint256) (contracts/vaults/DnGmxSeniorVault.sol#209-214) ignores return value by aUsdc.transferFrom(msg.sender,address(this),amount) (contracts/vaults/DnGmxSeniorVault.sol#213)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

StableSwapMock.exchange(uint256,uint256,uint256,uint256,bool) (contracts/mocks/StableSwapMock.sol#26-43) ignores return value by coins[i].transferFrom(msg.sender,address(this),dx) (contracts/mocks/
StableSwapMock.sol#36)
StableSwapMock.exchange(uint256,uint256,uint256,uint256,bool) (contracts/mocks/StableSwapMock.sol#26-43) ignores return value by coins[j].transfer(msg.sender,dy) (contracts/mocks/StableSwapMock.sol#42)
DepositPeriphery.depositToken(address,address,uint256) (contracts/periphery/DepositPeriphery.sol#100-112) ignores return value by IERC20(token).transferFrom(msg.sender,address(this),tokenAmount) (contracts/
periphery/DepositPeriphery.sol#105)
DnGmxBatchingManager.depositToken(address,uint256,uint256) (contracts/vaults/DnGmxBatchingManager.sol#196-213) ignores return value by IERC20(token).transferFrom(msg.sender,address(this),amount) (contracts/
vaults/DnGmxBatchingManager.sol#206)
DnGmxBatchingManager.depositUsdc(uint256,address) (contracts/vaults/DnGmxBatchingManager.sol#215-245) ignores return value by usdc.transferFrom(msg.sender,address(this),amount) (contracts/vaults/
DnGmxBatchingManager.sol#222)
DnGmxBatchingManager.executeBatchDeposit(uint256) (contracts/vaults/DnGmxBatchingManager.sol#260-275) ignores return value by sGlp.transfer(address(dnGmxJuniorVault),glpToTransfer) (contracts/vaults/
DnGmxBatchingManager.sol#268)
DnGmxBatchingManager._executeVaultUserBatchDeposit(uint256) (contracts/vaults/DnGmxBatchingManager.sol#389-431) ignores return value by sGlp.transfer(address(bypass),sGlpToDeposit) (contracts/vaults/
DnGmxBatchingManager.sol#404)
DnGmxBatchingManager._claim(address,address,uint256) (contracts/vaults/DnGmxBatchingManager.sol#433-466) ignores return value by dnGmxJuniorVault.transfer(receiver,amount) (contracts/vaults/
DnGmxBatchingManager.sol#463)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer      You, 1 second ago • Uncommitted changes
```

AUTOMATED TESTS (SLITHER)

```
..... LOW LEVEL-WARNINGS .....

FixedPointMathLib.rpow(uint256,uint256,uint256) (node_modules/@rari-capital/solmate/src/utils/FixedPointMathLib.sol#74-160) performs a multiplication on the result of a division:
    - x = xxRound_rpow_asm_0 / scalar (node_modules/@rari-capital/solmate/src/utils/FixedPointMathLib.sol#131)
    - zx_rpow_asm_0 = z * x (node_modules/@rari-capital/solmate/src/utils/FixedPointMathLib.sol#136)
FullMath.mulDiv(uint256,uint256,uint256) (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#14-108) performs a multiplication on the result of a division:
    - denominator = denominator / twos (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#68)
    - inv = (3 * denominator) ^ 2 (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#88)
FullMath.mulDiv(uint256,uint256,uint256) (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#14-108) performs a multiplication on the result of a division:
    - denominator = denominator / twos (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#68)
    - inv *= 2 - denominator * inv (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#92)
FullMath.mulDiv(uint256,uint256,uint256) (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#14-108) performs a multiplication on the result of a division:
    - denominator = denominator / twos (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#68)
    - inv *= 2 - denominator * inv (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#93)
FullMath.mulDiv(uint256,uint256,uint256) (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#14-108) performs a multiplication on the result of a division:
    - denominator = denominator / twos (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#68)
    - inv *= 2 - denominator * inv (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#94)
FullMath.mulDiv(uint256,uint256,uint256) (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#14-108) performs a multiplication on the result of a division:
    - denominator = denominator / twos (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#68)
    - inv *= 2 - denominator * inv (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#95)
FullMath.mulDiv(uint256,uint256,uint256) (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#14-108) performs a multiplication on the result of a division:
    - denominator = denominator / twos (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#68)
    - inv *= 2 - denominator * inv (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#96)
FullMath.mulDiv(uint256,uint256,uint256) (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#14-108) performs a multiplication on the result of a division:
    - denominator = denominator / twos (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#68)
    - inv *= 2 - denominator * inv (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#97)
FullMath.mulDiv(uint256,uint256,uint256) (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#14-108) performs a multiplication on the result of a division:
    - prod0 = prod0 / twos (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#73)
    - result = prod0 * inv (node_modules/@uniswap/v3-core-0.8-support/contracts/libraries/FullMath.sol#105)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

DnGmxJuniorVault.convertToAssets(uint256) (contracts/vaults/DnGmxJuniorVault.sol#567-571) uses a dangerous strict equality:
    - supply == 0 (contracts/vaults/DnGmxJuniorVault.sol#570)
DnGmxJuniorVault.convertToShares(uint256) (contracts/vaults/DnGmxJuniorVault.sol#558-562) uses a dangerous strict equality:
    - supply == 0 (contracts/vaults/DnGmxJuniorVault.sol#561)
DnGmxJuniorVault.previewMint(uint256) (contracts/vaults/DnGmxJuniorVault.sol#576-580) uses a dangerous strict equality:
    - supply == 0 (contracts/vaults/DnGmxJuniorVault.sol#579)
DnGmxJuniorVault.previewRedeem(uint256) (contracts/vaults/DnGmxJuniorVault.sol#597-604) uses a dangerous strict equality:
    - supply == 0 (contracts/vaults/DnGmxJuniorVault.sol#600-603)
DnGmxJuniorVault.previewWithdraw(uint256) (contracts/vaults/DnGmxJuniorVault.sol#585-592) uses a dangerous strict equality:
    - supply == 0 (contracts/vaults/DnGmxJuniorVault.sol#588-591)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

DnGmxBatchingManager.claimAndRedeem(address) (contracts/vaults/DnGmxBatchingManager.sol#284-296) uses a dangerous strict equality:
    - shares == 0 (contracts/vaults/DnGmxBatchingManager.sol#289)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

Reentrancy in ERC4626Upgradeable.deposit(uint256,address) (contracts/ERC4626/ERC4626Upgradeable.sol#59-71):      You, 11 seconds ago • Uncommitted changes
    External calls:
    - IERC20Metadata(asset).safeTransferFrom(msg.sender,address(this),assets) (contracts/ERC4626/ERC4626Upgradeable.sol#64)
    State variables written after the call(s):
    - _mint(receiver,shares) (contracts/ERC4626/ERC4626Upgradeable.sol#66)
        - _totalSupply += amount (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#269)
Reentrancy in ERC4626Upgradeable.mint(uint256,address) (contracts/ERC4626/ERC4626Upgradeable.sol#84-95):
    External calls:
    - IERC20Metadata(asset).safeTransferFrom(msg.sender,address(this),assets) (contracts/ERC4626/ERC4626Upgradeable.sol#88)
    State variables written after the call(s):
    - _mint(receiver,shares) (contracts/ERC4626/ERC4626Upgradeable.sol#90)
        - _totalSupply += amount (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#269)
Reentrancy in DnGmxJuniorVault.rebalance() (contracts/vaults/DnGmxJuniorVault.sol#373-392):
    External calls:
    - state.harvestFees() (contracts/vaults/DnGmxJuniorVault.sol#377)
    - state.rebalanceProfit(totalCurrentBorrowValue) (contracts/vaults/DnGmxJuniorVault.sol#383)
    - isPartialHedge = state.rebalanceHedge(currentBtc,currentEth,totalAssets(),true) (contracts/vaults/DnGmxJuniorVault.sol#388)
    State variables written after the call(s):
    - state.lastRebalanceTS = uint48(block.timestamp) (contracts/vaults/DnGmxJuniorVault.sol#390)
Reentrancy in DnGmxJuniorVault.stopVestAndStakeEsGmx() (contracts/vaults/DnGmxJuniorVault.sol#338-345):
    External calls:
    - IVester(state.rewardRouter.glpVester()).withdraw() (contracts/vaults/DnGmxJuniorVault.sol#341)
    - state.rewardRouter.stakeEsGmx(esGmxWithdrawn) (contracts/vaults/DnGmxJuniorVault.sol#343)
    State variables written after the call(s):
    - state.protocolEsGmx += esGmxWithdrawn (contracts/vaults/DnGmxJuniorVault.sol#344)
Reentrancy in DnGmxJuniorVault.unstakeAndVestEsGmx() (contracts/vaults/DnGmxJuniorVault.sol#327-334):
    External calls:
    - state.rewardRouter.unstakeEsGmx(state.protocolEsGmx) (contracts/vaults/DnGmxJuniorVault.sol#331)
    - IVester(state.rewardRouter.glpVester()).deposit(state.protocolEsGmx) (contracts/vaults/DnGmxJuniorVault.sol#332)
    State variables written after the call(s):
    - state.protocolEsGmx = 0 (contracts/vaults/DnGmxJuniorVault.sol#333)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
```

AUTOMATED TESTS (SLITHER)

```
Reentrancy in DnGmxBatchingManager._executeVaultUserBatchDeposit(uint256) (contracts/vaults/DnGmxBatchingManager.sol#389-431):
    External calls:
    - sGlp.transfer(address(bypass),sGlpToDeposit) (contracts/vaults/DnGmxBatchingManager.sol#404)
    - totalShares = bypass.deposit(sGlpToDeposit,address(this)) (contracts/vaults/DnGmxBatchingManager.sol#405)
    State variables written after the call(s):
    - vaultBatchingState.roundSharesMinted += totalShares (contracts/vaults/DnGmxBatchingManager.sol#407)
    - vaultBatchingState.roundDeposits[vaultBatchingState.currentRound] = RoundDeposit(vaultBatchingState.roundUsdcBalance.toUint128(),vaultBatchingState.roundSharesMinted.toUint128()) (contracts/vaults/DnGmxBatchingManager.sol#413-416)
    - vaultBatchingState.roundUsdcBalance = 0 (contracts/vaults/DnGmxBatchingManager.sol#426)
    - vaultBatchingState.roundGlpStaked = 0 (contracts/vaults/DnGmxBatchingManager.sol#427)
    - vaultBatchingState.roundSharesMinted = 0 (contracts/vaults/DnGmxBatchingManager.sol#428)
    - ++ vaultBatchingState.currentRound (contracts/vaults/DnGmxBatchingManager.sol#429)
Reentrancy in DnGmxBatchingManager._executeVaultUserBatchStake() (contracts/vaults/DnGmxBatchingManager.sol#370-387):
    External calls:
    - _roundGlpStaked = _stakeGlp(address(usdc),_roundUsdcBalance,minUsdg) (contracts/vaults/DnGmxBatchingManager.sol#381)
        - IERC20(token).approve(address(glpManager),amount) (contracts/vaults/DnGmxBatchingManager.sol#365)
        - glpStaked = rewardRouter.mintAndStakeGlp(token,amount,minUSDG,0) (contracts/vaults/DnGmxBatchingManager.sol#367)
    State variables written after the call(s):
    - vaultBatchingState.roundGlpStaked = _roundGlpStaked (contracts/vaults/DnGmxBatchingManager.sol#383)
    - vaultBatchingState.roundGlpDepositPending = _roundGlpStaked (contracts/vaults/DnGmxBatchingManager.sol#384)
Reentrancy in DnGmxBatchingManager.depositUsdc(uint256,address) (contracts/vaults/DnGmxBatchingManager.sol#215-245):
    External calls:
    - usdc.transferFrom(msg.sender,address(this),amount) (contracts/vaults/DnGmxBatchingManager.sol#222)
    State variables written after the call(s):
    - userDeposit.unclaimedShares += userDeposit.usdcBalance.mulDiv(roundDeposit.totalShares,roundDeposit.totalUsdc).toUint128() (contracts/vaults/DnGmxBatchingManager.sol#232-235)
    - userDeposit.round = vaultBatchingState.currentRound (contracts/vaults/DnGmxBatchingManager.sol#240)
    - userDeposit.usdcBalance = userUsdcBalance + amount.toUint128() (contracts/vaults/DnGmxBatchingManager.sol#241)
    - vaultBatchingState.roundUsdcBalance += amount.toUint128() (contracts/vaults/DnGmxBatchingManager.sol#242)
Reentrancy in DnGmxBatchingManager.executeBatchDeposit(uint256) (contracts/vaults/DnGmxBatchingManager.sol#260-275):
    External calls:
    - sGlp.transfer(address(dnGmxJuniorVault),glpToTransfer) (contracts/vaults/DnGmxBatchingManager.sol#268)
    - _executeVaultUserBatchDeposit(depositAmount) (contracts/vaults/DnGmxBatchingManager.sol#272)
        - sGlp.transfer(address(bypass),sGlpToDeposit) (contracts/vaults/DnGmxBatchingManager.sol#404)
        - totalShares = bypass.deposit(sGlpToDeposit,address(this)) (contracts/vaults/DnGmxBatchingManager.sol#405)
    State variables written after the call(s):
    - _pause() (contracts/vaults/DnGmxBatchingManager.sol#274)
        - _paused = true (node_modules/@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol#95)
Reentrancy in DnGmxBatchingManager.executeBatchStake() (contracts/vaults/DnGmxBatchingManager.sol#248-257):
    External calls:
    - dnGmxJuniorVault.harvestFees() (contracts/vaults/DnGmxBatchingManager.sol#250)
    - _executeVaultUserBatchStake() (contracts/vaults/DnGmxBatchingManager.sol#253)
        - IERC20(token).approve(address(glpManager),amount) (contracts/vaults/DnGmxBatchingManager.sol#365)
        - glpStaked = rewardRouter.mintAndStakeGlp(token,amount,minUSDG,0) (contracts/vaults/DnGmxBatchingManager.sol#367)
    State variables written after the call(s):
    - _pause() (contracts/vaults/DnGmxBatchingManager.sol#256)
        - _paused = true (node_modules/@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol#95)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

DnGmxJuniorVaultManager.receiveFlashLoan(DnGmxJuniorVaultManager.State,IERC20[],uint256[],bytes).btcAssetPremium (contracts/libraries/DnGmxJuniorVaultManager.sol#762) is a local variable never initialized
DnGmxJuniorVaultManager.receiveFlashLoan(DnGmxJuniorVaultManager.State,IERC20[],uint256[],bytes).ethAssetPremium (contracts/libraries/DnGmxJuniorVaultManager.sol#763) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

DnGmxJuniorVaultManager._executeRepay(DnGmxJuniorVaultManager.State,address,uint256) (contracts/libraries/DnGmxJuniorVaultManager.sol#835-841) ignores return value by state.pool.repay(token,amount,VARIABLE_INTEREST_MODE,address(this)) (contracts/libraries/DnGmxJuniorVaultManager.sol#840)
DnGmxJuniorVaultManager._executeWithdraw(DnGmxJuniorVaultManager.State,address,uint256,address) (contracts/libraries/DnGmxJuniorVaultManager.sol#860-867) ignores return value by state.pool.withdraw(token,amount,receiver) (contracts/libraries/DnGmxJuniorVaultManager.sol#866)
DnGmxJuniorVaultMock.executeRepay(address,uint256) (contracts/mocks/DnGmxJuniorVaultMock.sol#111-113) ignores return value by state.pool.repay(token,amount,VARIABLE_INTEREST_MODE,address(this)) (contracts/mocks/DnGmxJuniorVaultMock.sol#112)
DnGmxJuniorVaultMock.executeWithdraw(address,uint256) (contracts/mocks/DnGmxJuniorVaultMock.sol#119-121) ignores return value by state.pool.withdraw(token,amount,address(this)) (contracts/mocks/DnGmxJuniorVaultMock.sol#120)
DnGmxJuniorVaultMock.depositToken(address,uint256,uint256) (contracts/mocks/DnGmxJuniorVaultMock.sol#222-228) ignores return value by state.batchingManager.depositToken(token,amount,minUsdg) (contracts/mocks/DnGmxJuniorVaultMock.sol#227)
DnGmxJuniorVault.grantAllowances() (contracts/vaults/DnGmxJuniorVault.sol#134-164) ignores return value by state.wbtc.approve(aavePool,type()(uint256).max) (contracts/vaults/DnGmxJuniorVault.sol#139)
DnGmxJuniorVault.grantAllowances() (contracts/vaults/DnGmxJuniorVault.sol#134-164) ignores return value by state.wbtc.approve(swapRouter,type()(uint256).max) (contracts/vaults/DnGmxJuniorVault.sol#141)
DnGmxJuniorVault.grantAllowances() (contracts/vaults/DnGmxJuniorVault.sol#134-164) ignores return value by state.weth.approve(aavePool,type()(uint256).max) (contracts/vaults/DnGmxJuniorVault.sol#144)
DnGmxJuniorVault.grantAllowances() (contracts/vaults/DnGmxJuniorVault.sol#134-164) ignores return value by state.weth.approve(swapRouter,type()(uint256).max) (contracts/vaults/DnGmxJuniorVault.sol#146)
DnGmxJuniorVault.grantAllowances() (contracts/vaults/DnGmxJuniorVault.sol#134-164) ignores return value by state.weth.approve(address(state.batchingManager),type()(uint256).max) (contracts/vaults/DnGmxJuniorVault.sol#148)
DnGmxJuniorVault.grantAllowances() (contracts/vaults/DnGmxJuniorVault.sol#134-164) ignores return value by state.usdc.approve(aavePool,type()(uint256).max) (contracts/vaults/DnGmxJuniorVault.sol#151)
DnGmxJuniorVault.grantAllowances() (contracts/vaults/DnGmxJuniorVault.sol#134-164) ignores return value by state.usdc.approve(address(swapRouter),type()(uint256).max) (contracts/vaults/DnGmxJuniorVault.sol#153)
DnGmxJuniorVault.grantAllowances() (contracts/vaults/DnGmxJuniorVault.sol#134-164) ignores return value by state.usdc.approve(address(state.batchingManager),type()(uint256).max) (contracts/vaults/DnGmxJuniorVault.sol#155)
DnGmxJuniorVault.grantAllowances() (contracts/vaults/DnGmxJuniorVault.sol#134-164) ignores return value by state.aUsdc.approve(address(state.dnGmxSeniorVault),type()(uint256).max) (contracts/vaults/DnGmxJuniorVault.sol#158)
DnGmxJuniorVault.grantAllowances() (contracts/vaults/DnGmxJuniorVault.sol#134-164) ignores return value by IERC20Metadata(asset).approve(address(state.glpManager),type()(uint256).max) (contracts/vaults/DnGmxJuniorVault.sol#161)
DnGmxJuniorVault.beforeWithdraw(uint256,uint256,address) (contracts/vaults/DnGmxJuniorVault.sol#729-738) ignores return value by state.rebalanceHedge(currentBtc,currentEth,totalAssets() - assets,false) (contracts/vaults/DnGmxJuniorVault.sol#737)
DnGmxJuniorVault.afterDeposit(uint256,uint256,address) (contracts/vaults/DnGmxJuniorVault.sol#740-750) ignores return value by state.rebalanceHedge(currentBtc,currentEth,totalAssets(),false) (contracts/vaults/DnGmxJuniorVault.sol#749)
DnGmxSeniorVault.initialize(address,string,string,address) (contracts/vaults/DnGmxSeniorVault.sol#88-98) ignores return value by aUsdc.approve(address(pool),type()(uint256).max) (contracts/vaults/DnGmxSeniorVault.sol#96)
DnGmxSeniorVault.initialize(address,string,string,address) (contracts/vaults/DnGmxSeniorVault.sol#88-98) ignores return value by IERC20(asset).approve(address(pool),type()(uint256).max) (contracts/vaults/DnGmxSeniorVault.sol#97)
DnGmxSeniorVault.grantAllowances() (contracts/vaults/DnGmxSeniorVault.sol#102-111) ignores return value by IERC20(asset).approve(aavePool,type()(uint256).max) (contracts/vaults/DnGmxSeniorVault.sol#106)
DnGmxSeniorVault.grantAllowances() (contracts/vaults/DnGmxSeniorVault.sol#102-111) ignores return value by aUsdc.approve(aavePool,type()(uint256).max) (contracts/vaults/DnGmxSeniorVault.sol#108)
DnGmxSeniorVault.updateBorrowCap(address,uint256) (contracts/vaults/DnGmxSeniorVault.sol#166-177) ignores return value by aUsdc.approve(borrowerAddress,cap) (contracts/vaults/DnGmxSeniorVault.sol#174)
DnGmxSeniorVault.beforeWithdraw(uint256,uint256,address) (contracts/vaults/DnGmxSeniorVault.sol#288-300) ignores return value by pool.withdraw(address(asset),assets,address(this)) (contracts/vaults/DnGmxSeniorVault.sol#299)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

BatchingManagerBypass.setSglp(IERC20) (contracts/periphery/BatchingManagerBypass.sol#27-30) ignores return value by sGlp.approve(address(juniorVault),type()(uint256).max) (contracts/periphery/BatchingManagerBypass.sol#29)
DepositPeriphery.setAddresses(IDnGmxJuniorVault,IRewardRouterV2,IGlpManager) (contracts/periphery/DepositPeriphery.sol#73-93) ignores return value by sGlp.approve(address(_dnGmxJuniorVault),type()(uint256).max) (contracts/periphery/DepositPeriphery.sol#90)
DepositPeriphery._convertToSglp(address,uint256) (contracts/periphery/DepositPeriphery.sol#114-129) ignores return value by IERC20(token).approve(address(glpManager),tokenAmount) (contracts/periphery/DepositPeriphery.sol#115)
WithdrawPeriphery.setAddresses(IDnGmxJuniorVault,IRewardRouterV2) (contracts/periphery/WithdrawPeriphery.sol#86-108) ignores return value by sGlp.approve(address(glpManager),type()(uint256).max) (contracts/periphery/WithdrawPeriphery.sol#105)
DnGmxBatchingManager.grantAllowances() (contracts/vaults/DnGmxBatchingManager.sol#145-147) ignores return value by sGlp.approve(address(dnGmxJuniorVault),type()(uint256).max) (contracts/vaults/DnGmxBatchingManager.sol#146)
DnGmxBatchingManager._stakeGlp(address,uint256,uint256) (contracts/vaults/DnGmxBatchingManager.sol#359-368) ignores return value by IERC20(token).approve(address(glpManager),amount) (contracts/vaults/DnGmxBatchingManager.sol#365)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

AUTOMATED TESTS RESULTS

While reviewing the contract code with the Slither tool, we identified some false positive errors that were reported, which are added above. We want to note that we have properly examined all other issues that were identified, and we are confident that the code is free of bugs. Specifically, some of the false positives were related to returning values from external checks, which do not pose a security risk to the smart contract.

ABOUT FORTITUDE

Fortitude is founded in 2021, which aims to provide reliable and trustworthy crypto audits to ensure the stability and security of the cryptocurrency market.

By to date, Vaultka have performed audit for more than 50 projects with over 70k of code, secured $75M USD.

Visit us: https://www.fortitudeaudit.com/

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice unless indicated otherwise by Fortitude; however, Fortitude does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Fortitude.

Links to other websites

You may, through hypertext or other computer links, gain access to websites operated by persons other than Fortitude. Such hyperlinks are provided for your reference and convenience only and are the exclusive responsibility of such web sites&aspo; owners. You agree that Fortitude is not responsible for the content or operation of such websites and that Fortitude shall have no liability to you or any other person or entity for the use of third-party websites. Except as described below, a hyperlink from this website to another website does not imply or mean that Fortitude endorses the content on that website or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other websites to which you link from the report. Fortitude assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Fortitude disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, any other areas beyond the programming language, or other programming aspects that could present security risks. The report does not indicate the endorsement by Fortitude of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service, or any other asset. Fortitude does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by, or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.