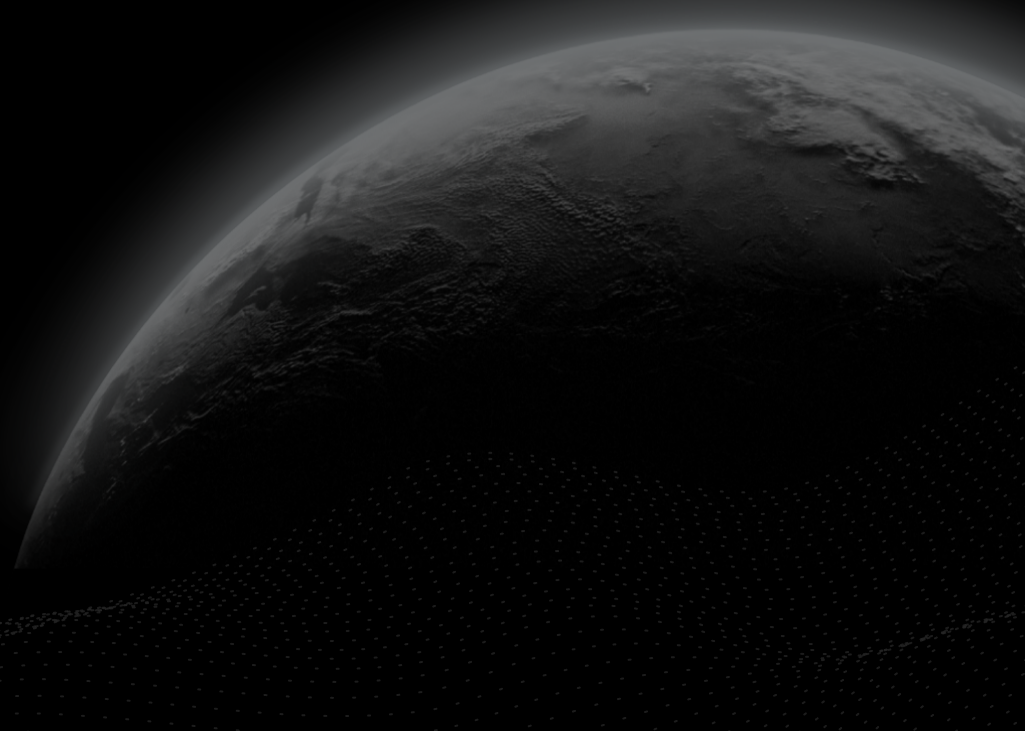
CERTIK

Security Assessment

Neutra - audit

CertiK Verified on Apr 10th, 2023



CertiK Verified on Apr 10th, 2023

Neutra - audit

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Others	Arbitrum	Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 04/10/2023	N/A

CODEBASE
https://github.com/NeutraFinance/neutra-gmx-contracts/tree/aba8d9e4dc49a37dab517baecf46663e26cf1c35
...View All

COMMITS
aba8d9e4dc49a37dab517baecf46663e26cf1c35
...View All

Vulnerability Summary

25	12	0	0	13	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

0	Critical		Critical 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.
2	Major	2 Resolved	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
8	Medium	4 Resolved, 4 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
15	Minor	6 Resolved, 9 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
0	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.

TABLE OF CONTENTS | NEUTRA - AUDIT

CODEBASE | NEUTRA - AUDIT

Repository

https://github.com/NeutraFinance/neutra-gmx-contracts/tree/aba8d9e4dc49a37dab517baecf46663e26cf1c35

Commit

aba8d9e4dc49a37dab517baecf46663e26cf1c35

AUDIT SCOPE | NEUTRA - AUDIT

7 files audited ● 7 files without findings

ID	File	SHA256 Checksum
BRN	contracts/BatchRouter.sol	6e7fccd8193533bd367d0e8de70f0c5da4ef1407d3bf02d19c6dc85e3d66c557
GHN	contracts/GmxHelper.sol	18148cff3f991d1f204924f7263b57ecaf6925843f3389d666dfc34853e764aa
ECT	contracts/ExecutionCallbackTarget.sol	8ed81a64f3b9a3fde331b4b89293da78ed94a454fe0cacaeb3275e8954f638d6
RCT	contracts/RepayCallbackTarget.sol	5bae60e1e73e7d291d46e84a84f5113574611d295a461e0cd25851f724843826
RVN	contracts/RouterV2.sol	36d030b3cb1e676d7df289d8517f9b8cc600529c85756f77b28e7769b6c12a33
SVV	contracts/StrategyVaultV2.sol	1e34ce0b8c81a129fbd9cf39df44795355881c84275776c6bdcc07bb728c85cd
SVN	contracts/StrategyVault.sol	1c4b8da9e20ee7a101df490c6c0b3a11789ad3ad0516c5ca32891f52914e21b5

APPROACH & METHODS | NEUTRA - AUDIT

This report has been prepared for Neutra to discover issues and vulnerabilities in the source code of the Neutra - audit project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

REVIEW NOTES | NEUTRA - AUDIT

Neutra Finance aims to make risk-hedged, sustainable investment strategies are easily accessible for anyone, anywhere through automated strategy vaults. Neutra makes this process simple and easy so that anyone who wants to protect their funds and earn stable returns in any market condition can do so. Upon depositing their capital into vaults, users can earn APY above market standards on high-performing DeFi products while the strategy will do the rest, such as optimizing returns, rebalancing, and managing liquidation risk.

Financial Models

Financial models of blockchain protocols need to be resilient to attacks. They need to pass simulations and verifications to guarantee the security of the overall protocol.

As per the whitepaper, GLP Market Neutral Vault (nGLP Vault) simplifies this process for users through the unique rebalancing algorithm and enables users to easily and safely earn double-digit APY while protecting equity value. Rebalancing occurs based on 1)liquidation risk and 2)asset weight deviation within GLP, it is triggered when asset weight deviation + predicted price volatility exceeds a certain threshold.

The rebalancing algorithm and the rebalancing trigger mechanism are not in the scope of this audit.

Third-Party Dependencies

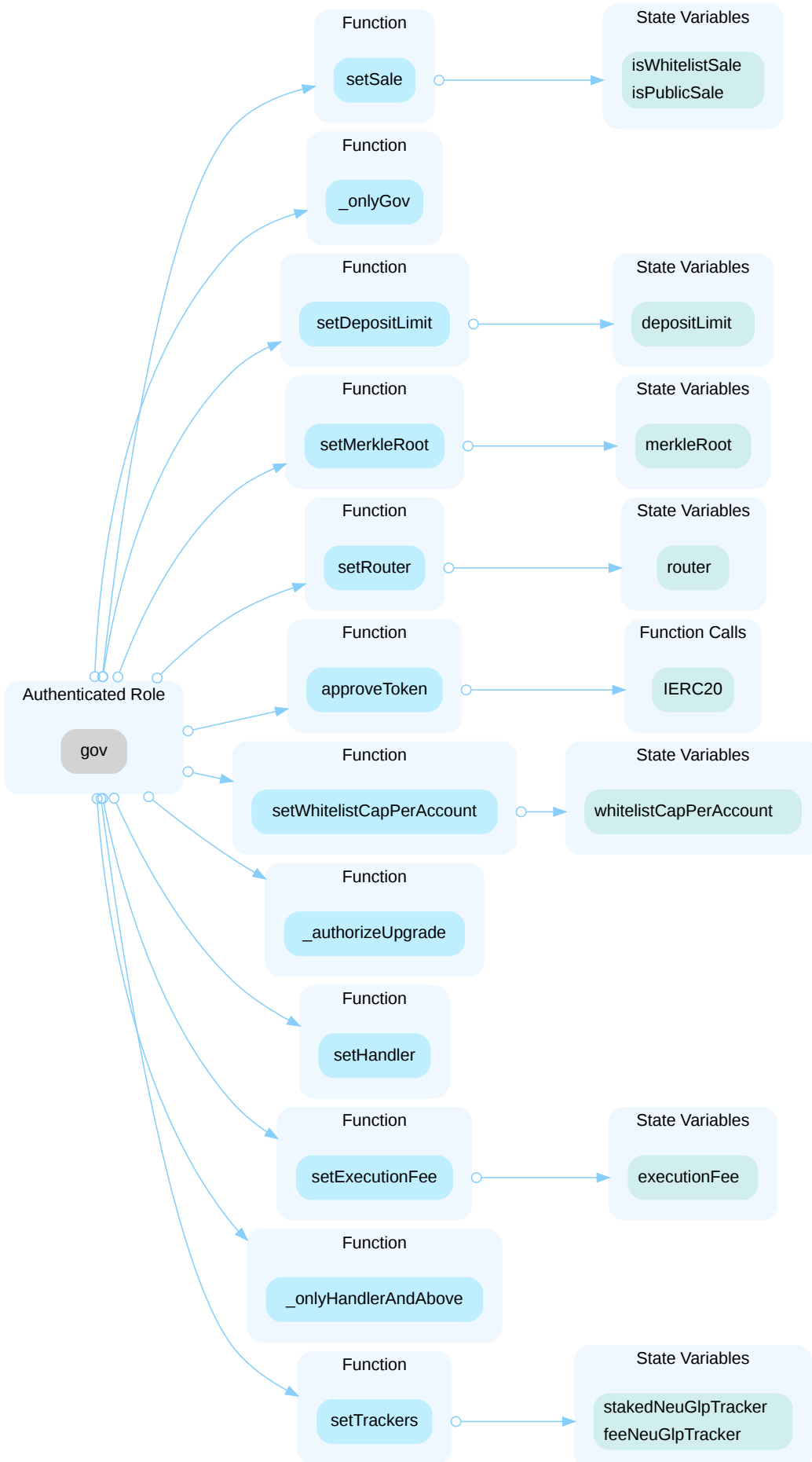
The contract serves as the underlying entity to interact with third-party protocols like `GMX`, etc. The scope of the audit treats 3rd party entities as black boxes and assumes their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

We understand that business logic requires interaction with `GMX`, etc. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

DECENTRALIZATION EFFORTS | NEUTRA - AUDIT

Description

In the contract `BatchRouter` the role `gov` has authority over the functions shown in the diagram below. Any compromise to the `gov` account may allow the hacker to take advantage of this authority.

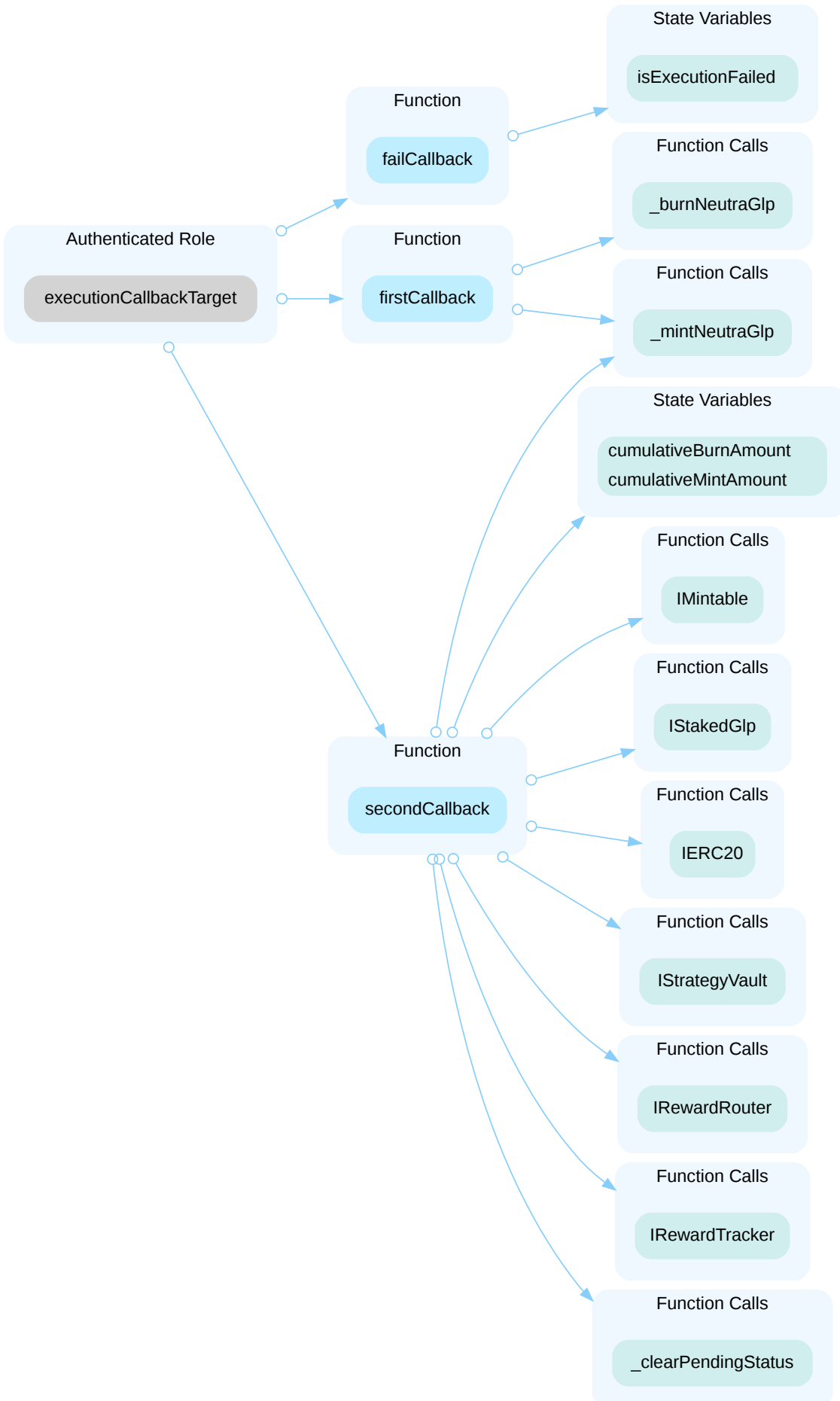

Function
setSale
State Variables
isWhitelistSale
isPublicSale
Function
_onlyGov
Function
setDepositLimit
State Variables
depositLimit
Function
setMerkleRoot
State Variables
merkleRoot
Function
setRouter
State Variables
router
Function
approveToken
Function Calls
IERC20
Authenticated Role
gov
Function
setWhitelistCapPerAccount
State Variables
whitelistCapPerAccount
Function
_authorizeUpgrade
Function
setHandler
Function
setExecutionFee
State Variables
executionFee
Function
_onlyHandlerAndAbove
Function
setTrackers
State Variables
stakedNeuGlpTracker
feeNeuGlpTracker

In the contract `BatchRouter` , the role `HandlerAndAbove` has authority over the following functions:

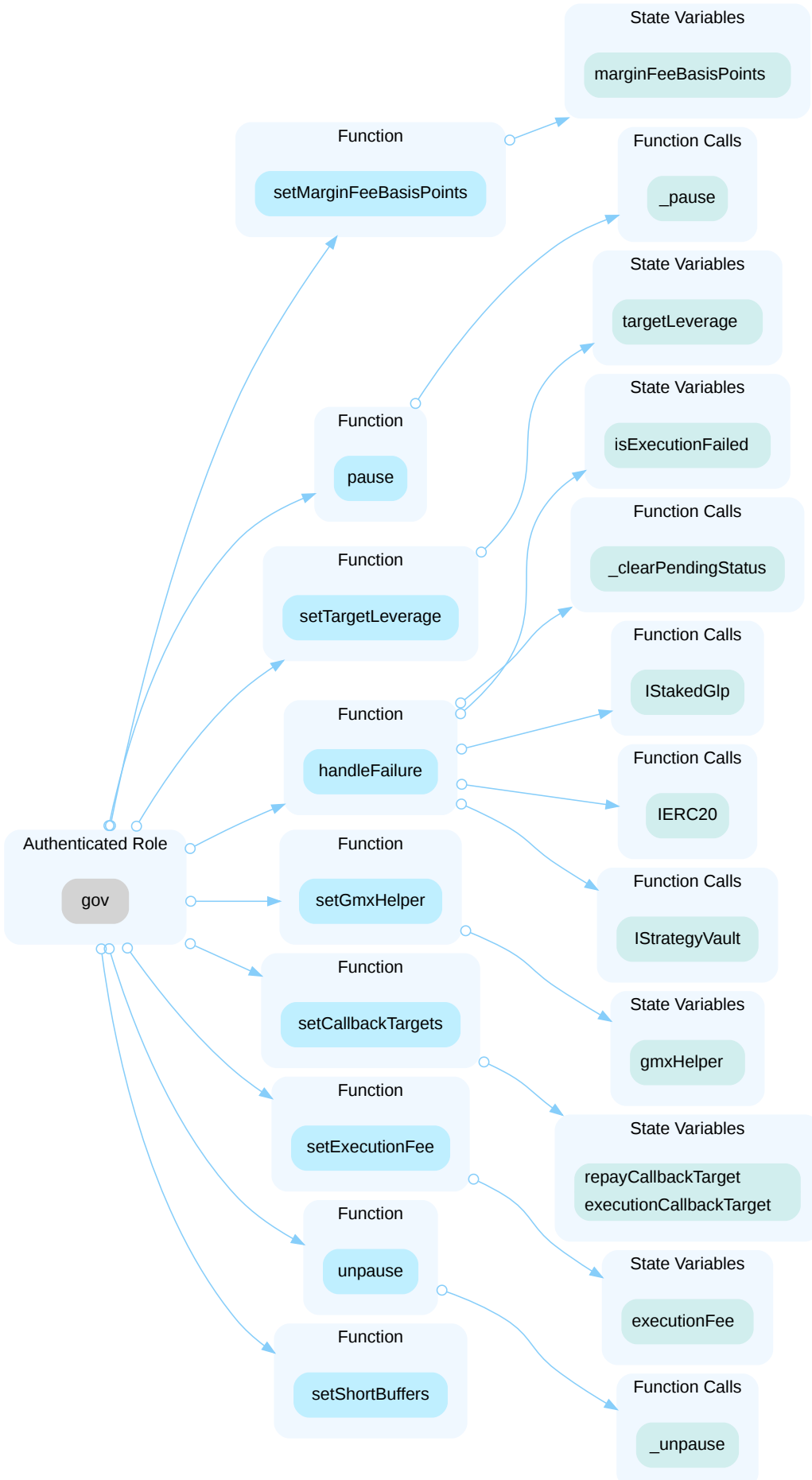
- function executeBatchPositions()
- function confirmAndDealGlp()

Any compromise to the `HandlerAndAbove` account may allow a hacker to take advantage of this authority.

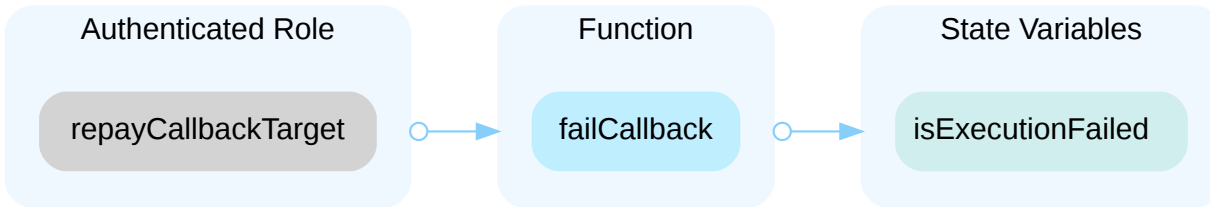
In the contract `RouterV2` the role `executionCallbackTarget` has authority over the functions shown in the diagram below. Any compromise to the `executionCallbackTarget` account may allow the hacker to take advantage of this authority.


State Variables
isExecutionFailed
Function
failCallback
Function Calls
_burnNeutraGlp
Authenticated Role
executionCallbackTarget
Function
firstCallback
Function Calls
_mintNeutraGlp
State Variables
cumulativeBurnAmount
cumulativeMintAmount
Function Calls
IMintable
Function Calls
IStakedGlp
Function
secondCallback
Function Calls
IERC20
Function Calls
IStrategyVault
Function Calls
IRewardRouter
Function Calls
IRewardTracker
Function Calls
_clearPendingStatus

In the contract `RouterV2` the role `gov` has authority over the functions shown in the diagram below. Any compromise to the `gov` account may allow the hacker to take advantage of this authority.


State Variables
marginFeeBasisPoints
Function
setMarginFeeBasisPoints
Function Calls
_pause
State Variables
targetLeverage
State Variables
isExecutionFailed
Function
pause
Function Calls
_clearPendingStatus
Function
setTargetLeverage
Function Calls
IStakedGlp
Function
handleFailure
Function Calls
IERC20
Function Calls
IStrategyVault
Authenticated Role
gov
Function
setGmxHelper
Function
setCallbackTargets
State Variables
gmxHelper
Function
setExecutionFee
State Variables
repayCallbackTarget
executionCallbackTarget
Function
unpause
State Variables
executionFee
Function
setShortBuffers
Function Calls
_unpause

In the contract `RouterV2` the role `repayCallbackTarget` has authority over the functions shown in the diagram below. Any compromise to the `repayCallbackTarget` account may allow the hacker to take advantage of this authority.



In the contract `StrategyVault`, the role `gov` has authority over the following functions:

- function activateManagementFee()
- function deactivateManagementFee()
- function exitStrategy()
- function depositInsuranceFund()
- function setGov()
- function setGmxHelper()
- function setMarginFeeBasisPoints()
- function setKeeper()
- function setWant()
- function setExecutionFee()
- function setCallbackTarget()
- function setRouter()
- function setManagement()
- function registerAndSetReferralCode()
- function withdrawFees()
- function withdrawInsuranceFund()
- function withdrawEth()
- function adjustPrepaidGmxFee()

Any compromise to the `gov` account may allow a hacker to take advantage of this authority.

In the contract `StrategyVault`, the role `router` has authority over the following functions:

- function settle()

Any compromise to the `router` account may allow the hacker to take advantage of this authority.

Any compromise to the `gov` account may allow a hacker to take advantage of this authority.

In the contract `StrategyVault`, the role `keepersAndAbove` has authority over the following functions:

- function minimiseDeltaWithBuyGlp()

- function retryPositions()
- function confirmRebalance()
- function repayFundingFee()
- function buyGlp()
- function sellGlp()
- function increaseShortPosition()
- function decreaseShortPosition()
- function repayUnpaidFundingFee()

Any compromise to the `keepersAndAbove` account may allow the hacker to take advantage of this authority.

In the contract `StrategyVault2`, the role `router` has authority over the following functions:

- function increaseShortPositionsWithCallback()
- function decreaseShortPositionsWithCallback()

Any compromise to the `router` account may allow the hacker to take advantage of this authority.

In the contract `StrategyVault2`, the role `gov` has authority over the following functions:

- function approveToken()

Any compromise to the `gov` account may allow the hacker to take advantage of this authority.

In the contract `StrategyVault2`, the role `keepersAndAbove` has authority over the following functions:

- function instantRepayFundingFee()
- function emergencyConfrim()

Any compromise to the `keepersAndAbove` account may allow the hacker to take advantage of this authority.

`BatchRouter`, `StrategyVault`, and `StrategyVault2` are upgradeable contracts, the owner can upgrade the contract without the community's commitment. If an attacker compromises the account, he can change the implementation of the contract and drain tokens from the contract.

Recommendations

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We recommend carefully managing the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

FINDINGS | NEUTRA - AUDIT

25	0	2	8	15	0
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Neutra - audit. Through this audit, we have uncovered 25 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
NFB-04	Potential Loss Of Escrowed Assets	Logical Issue	Major	Resolved
RVN-01	Potential Neutra Service Stuck	Logical Issue	Major	Resolved
BRN-01	Redundant Execution Fee Is Charged	Logical Issue	Medium	Acknowledged
BRN-02	Lack Of Validation Of `executed`	Logical Issue	Medium	Acknowledged
NFB-05	Functions Being Called Are Not Implemented	Logical Issue	Medium	Acknowledged
NFB-06	Lack Of Access Control	Logical Issue	Medium	Resolved
NFB-07	Potential Reentrancy Attack	Volatile Code	Medium	Resolved
NFB-09	Status Is Not Reset To `PositionExecutionStatus.NONE`	Logical Issue	Medium	Resolved
NFB-10	Potential Damage Due To Unprotected Initializer	Logical Issue	Medium	Resolved
RVN-02	Potential Unsuccessful Assets Withdraw	Logical Issue	Medium	Acknowledged
BRN-03	Claimable `want` And `snGlp` Depend On `_dealAmount`	Logical Issue	Minor	Acknowledged

ID	Title	Category	Severity	Status
NFB-08	Missing Add Losses From Shorts	Mathematical Operations	Minor	Resolved
NFB-11	Lack Of Reasonable Boundary	Logical Issue	Minor	Acknowledged
NFB-12	Missing Zero Address Validation	Volatile Code	Minor	Resolved
NFB-13	Unused Return Value	Volatile Code	Minor	Acknowledged
NFB-14	Unchecked ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	Acknowledged
NFB-15	Divide Before Multiply	Mathematical Operations	Minor	Acknowledged
NFU-05	Incompatibility With Deflationary Tokens	Logical Issue	Minor	Acknowledged
RVN-03	Missing Refund The Execution Fee Leftover	Logical Issue	Minor	Resolved
RVN-04	Constant Cumulative Funding Rate Update Interval	Logical Issue	Minor	Resolved
SVN-01	Lack Of Storage Gap In Upgradeable Contract	Logical Issue	Minor	Acknowledged
SVN-02	Lack Of Validation For Array Length	Logical Issue	Minor	Acknowledged
SVN-03	Lack Of Validation Of `confirmed`	Logical Issue	Minor	Resolved
SVN-04	Potential Pay Duplicated Unpaid Funding Fee	Logical Issue	Minor	Resolved
SVN-05	EOA Receive `want` When Selling `glp`	Logical Issue	Minor	Acknowledged

NFB-04 | POTENTIAL LOSS OF ESCROWED ASSETS

Category	Severity	Location	Status
Logical Issue	Major	contracts/ExecutionCallbackTarget.sol (Neutra): 27, 41; contracts/RouterV2.sol (Neutra): 220~226, 315, 337	Resolved

Description

When a user accesses the `instantDeposit()` or `instantDepositGlp()` function of the `RouterV2` contract, assets are deposited into Neutra Finance and the user receives `snGlp` tokens as proof of their share, which can be used for identity verification when redeeming the assets.

However, when the `_isStaked` parameter of the `instantWithdraw()` function is set to False, the user's identity is not verified when redeeming the assets, and the `_amount` parameter value is treated as the quantity of `nGlp` tokens. At this point, any user can call the `instantWithdraw()` function if there are `nGlp` tokens in the `RouterV2` contract, and receive the corresponding quantity of assets that other users have deposited into Neutra Finance, such as `DAI` or `fsGlp`.

Based on this code logic, there is a risk of front-running attacks and potential loss of assets deposited by users in Neutra Finance.

Scenario

Assumptions

1. Currently, a substantial amount of user assets are under custody by Neutra Finance, with a total issuance of `nGlp` amounting to `1083978e18`.
2. There is no need to repay funding fees when closing a position on GMX.
3. In a particular scenario, the Bob account on the Arbitrum One blockchain redeems `nGlp` tokens staked in the `StakedNeuGlpTracker` contract using the held `snGlp` tokens and receives `1000e18` `nGlp` tokens.
4. The fact that the Alice account on the Arbitrum One blockchain does not hold `snGlp` tokens indicates that she has not entrusted any assets to Neutra Finance.

Attack scenario

1. Bob plans to redeem custody assets corresponding to `100e18` nGlp tokens from Neutra Finance, so he transfers `100e18` nGlp tokens to the `RouterV2` contract.
2. At this point, Alice learns that the `nGlp` balance in the `RouterV2` contract is greater than zero through the following two methods:

   {a} Monitoring the `nGlp` token balance in the `RouterV2` contract.

{b} Monitoring the transaction information transferred to the `RouterV2` contract through the Arbitrum One blockchain transaction information cache pool.

3. After Bob transfers `100e18` nGlp tokens to the `RouterV2` contract, he calls the `instantWithdraw()` function to redeem the DAI assets.

4. At this point, Alice learns that Bob has called the `instantWithdraw()` function by monitoring the Arbitrum One blockchain transaction information cache pool. Alice pays more gas fees than Bob to ensure that the transaction that accesses the Neutra `RouterV2` contract's `instantWithdraw()` function with the following parameter values is executed first, and pays a GMX execution fee of `0.0004 ether`:

```
instantWithdraw(_amount=1000e18, _isStaked=False, _withdrawGlp=False);
```

5. The `_setPendingStatus()` function sets the `pendingStatus.recipient` status variable to Alice, indicating that Alice is the recipient of the DAI assets redeemed this time.

6. In the `instantWithdraw()` function, only lines 253-254 of code contain verification logic for `msg.sender`, that is, burning the `snGlp` tokens held by `msg.sender` to prove that `msg.sender` has invested assets in Neutra Finance. Since the parameter value `_isStaked=False`, lines 253-254 of code are not executed, allowing Alice to pass the verification successfully.

7. The following operations are all related to the assets of the `StrategyVaultV2` or `RouterV2` contract and have nothing to do with `msg.sender`, Alice. The management fee is collected by diluting the value of the `nGlp` token, regardless of whether `msg.sender` or Alice has custody assets.

   {a} `harvest()`, harvests `WETH` rewards from GMX and converts them to `DAI`.

   {b} `sellGlp()`, removes liquidity from GMX and retrieves `DAI` assets for users who are long on `WBTC` and `WETH`.

   {c} `_burnNeutraGlp()`, burns the `nGlp` tokens held by the `RouterV2` contract.

   {d} `decreaseShortPositionsWithCallback()`, creates a closing operation request for `WBTC` and `WETH` and retrieves `DAI` assets for those who are short on `WBTC` and `WETH` from GMX.

8. After Alice successfully calls the `instantWithdraw()` function, she waits for GMX to execute the closing operation for `WBTC` and `WETH`, and then calls back the `gmxPositionCallback()` function of the `ExecutionCallbackTarget` contract. At this time, Bob's transaction calling `instantWithdraw()` cannot be executed because other users are not allowed to execute the `instantWithdraw()` function when `pendingStatus.isProgress = true`.

9. Alice's identity is not verified during the execution of the callback function `gmxPositionCallback()`, and the retrieved `DAI` assets are directly transferred to `pendingStatus.recipient`, which is Alice.

10. In the end, Alice was able to obtain Bob's managed assets without investing any assets in Neutra Finance, simply by spending 0.0004 ether and paying for the gas fees to execute the `instantWithdraw()` function. Bob will lose the assets corresponding to 100e18 `nGlp` tokens.

Recommendation

We recommend refactoring the code to prevent the loss of user-hosted assets. Potential solutions are as follows:

1. Only permit users to redeem hosted assets using the `snGlp` function.
2. Record user information for assets hosted on Neutra Finance using state variables and validate users upon asset redemption.

Alleviation

`[CertiK]` : Neutra team has added code logic to allow only `nGlp/snGlp` holders to withdraw the escrowed assets and solved this issue in the commit ef9633222a5955081625f21f1d9aa8ed7b4a417d.

RVN-01 | POTENTIAL NEUTRA SERVICE STUCK

Category	Severity	Location	Status
Logical Issue	Major	contracts/RouterV2.sol (Neutra): 152, 161, 195, 442~445	Resolved

Description

The initial assets of the `StrategyVaultV2` contract of Neutra Finance are invested by users. Therefore, before the first user invests assets in Neutra Finance, the total assets in the `StrategyVaultV2` contract and the total issuance of the voucher token `nGlp` representing the aforementioned asset shares, are both zero.

At this point, when the first user deposits assets and purchases GMX `Glp`, they need to mint a corresponding amount of `nGlp`. However, the `_mintNeutraGlp()` function does not handle the special case where the state variables `pendingStatus.totalSupplyBefore` and `pendingStatus.totalValueBefore` are both zero, resulting in the rollback of the first user's asset deposit operation.

Scenario

Assuming the following conditions:

1. Neutra Finance currently has zero assets under management, so the total value of the `StrategyVaultV2` contract is zero, and the total issuance of `nGlp` tokens is zero.

Attack Scenario:

1. Bob is the first user of Neutra Finance, and he deposits 100 `DAI` assets from his holdings into Neutra Finance by calling the `instantDeposit()` function.
2. Neutra Finance divides the 100 `DAI` assets deposited by Bob into three parts, which are used to (1) buy GLP, (2) short WBTC, and (3) short WETH.
3. The `_setPendingStatus()` function is called on line 195, and afterwards, `pendingStatus.totalSupplyBefore` and `pendingStatus.totalValueBefore` are both zero.
4. After line 159 `uint256 amountOut = IStrategyVault(strategyVault).buyGlp(glpAmountIn);` is executed, the `StrategyVaultV2` contract holds `amountOut` amount of `fsGlp`.
5. `_mintNeutraGlp(0x0)` is called on line 161 to mint `nGlp` tokens.
6. Due to the fact that the state variables `pendingStatus.totalSupplyBefore` and `pendingStatus.totalValueBefore` are both zero, line 445 `mintAmount = increasedValue * pendingStatus.totalSupplyBefore / pendingStatus.totalValueBefore;` will fail.
7. This ultimately results in the rollback of Bob's asset deposit operation, and worst of all, the Neutra Finance project will be unable to function properly.

Recommendation

We recommend modifying the code to avoid the failure of the first user's asset deposit in Neutra Finance. One potential solution is to mint the number of `nGlp` tokens equal to the increase in the total value of the `StrategyVaultV2` contract when both the state variables `pendingStatus.totalSupplyBefore` and `pendingStatus.totalValueBefore` are zero.

Alleviation

`[CertiK]` : Neutra team has added code logic such that when the assets held in the `StrategyVaultV2` contract are zero, the amount of `nGlp` tokens minted is equal to the increased value in the contract. This issue is solved in commit ef9633222a5955081625f21f1d9aa8ed7b4a417d.

BRN-01 | REDUNDANT EXECUTION FEE IS CHARGED

Category	Severity	Location	Status
Logical Issue	Medium	contracts/BatchRouter.sol (Neutra): 309	Acknowledged

Description

The function `executeBatchPositions` will eventually call the function `executeIncreasePositions` of the strategy vault to increase WBTC/WETH position. However, the function `executeIncreasePositions` seems deprecated and is not implemented. So the execution fee should not be charged.

Recommendation

We recommend reviewing the logic again and fixing the issue.

Alleviation

`[Neutra]`: We will no longer use `BatchRouter` contract.

BRN-02 | LACK OF VALIDATION OF `executed`

Category	Severity	Location	Status
Logical Issue	Medium	contracts/BatchRouter.sol (Neutra): 306	Acknowledged

Description

There is no guarantee that batch is not under execution.

Recommendation

We recommend adding the validation as below.

```
require(!executed, "BatchRouter: batch under execution");
```

Alleviation

`[Neutra]`: We will no longer use `BatchRouter` contract.

NFB-05 | FUNCTIONS BEING CALLED ARE NOT IMPLEMENTED

Category	Severity	Location	Status
Logical Issue	Medium	contracts/BatchRouter.sol (Neutra): 309, 322, 337; contracts/StrategyVault.sol (Neutra): 325~349	Acknowledged

Description

The contract `BatchRouer` will eventually call the functions of the Strategy Vault are not implemented, which will impact the intended behavior of the whole contract.

- function executeBatchPositions()->IRouter(router).executePositionsBeforeDealGlp()->IStrategyVault(strategyVault).executeIncreasePositions()
- function confirmAndDealGlp()->IRouter(router).confirmAndBuy()->_vault.buyNeuGlp()
- function confirmAndDealGlp()->IRouter(router).confirmAndSell()->_vault.sellNeuGlp()

```
    /// deprecated
    /// @dev withdraw init function
    /// execute wbtc, weth decrease positions
    function executeDecreasePositions(bytes[] calldata _params) external payable
onlyRouter {
    }

    /// deprecated
    /// @dev should be called only if positions execution had been failed
    function retryPositions(bytes4[] calldata _selectors, bytes[] calldata _params)
external payable onlyKeepersAndAbove {
    }

    /// deprecated
    function buyNeuGlp(uint256 _amountIn) external onlyRouter returns (uint256) {
    }

    /// deprecated
    function sellNeuGlp(uint256 _glpAmount, address _recipient) external onlyRouter
returns (uint256) {
    }
```

Due to these unimplemented functions, users are unable to successfully buy `glp` and stake it to get rewards.

Recommendation

We recommend reviewing the logic and ensuring it is as intended. And we also recommend reviewing all the unimplemented functions marked with deprecated in the contract `StrategyVault` ensuring they are not referenced in the protocols.

Alleviation

`[Neutra]`: We will no longer use `BatchRouter` contract.

NFB-06 | LACK OF ACCESS CONTROL

Category	Severity	Location	Status
Logical Issue	Medium	contracts/ExecutionCallbackTarget.sol (Neutra): 27; contracts/RepayCallbackTarget.sol (Neutra): 22; contracts/StrategyVaultV2.sol (Neutra): 186	Resolved

Description

The function `gmxPositionCallback` can be called by anyone as it has no access restriction. This enables anyone to call this and control the position execution result.

The function `confirmCallback()` can be called by anyone as it has no access restriction.

Recommendation

Consider adding a modifier that who can call these functions.

Alleviation

`[CertiK]`: The team updated the code in commits ef9633222a5955081625f21f1d9aa8ed7b4a417d.

NFB-07 | POTENTIAL REENTRANCY ATTACK

Category	Severity	Location	Status
Volatile Code	Medium	contracts/BatchRouter.sol (Neutra): 219, 220, 223, 322, 324, 331, 334, 337, 342, 345, 349, 350; contracts/RouterV2.sol (Neutra): 149, 151, 152, 192, 194, 195, 247, 249, 250, 253, 254, 270, 271, 272, 319, 321, 324, 341, 346, 348, 349, 351, 354, 359, 360, 361, 447, 458, 469, 481, 483, 489, 504, 527, 528, 529, 530, 533, 536, 537; contracts/StrategyVault.sol (Neutra): 197, 226, 229, 249, 264, 294, 297, 317, 420, 429, 456, 457, 536, 637, 638, 639, 706, 709, 716	Resolved

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

External call(s)

```
219          IRewardTracker(feeNeuGlpTracker).stakeForAccount(address(this), msg.sender, nGlp, _amount);
```

```
220          IRewardTracker(stakedNeuGlpTracker).stakeForAccount(msg.sender, msg.sender, feeNeuGlpTracker, _amount);
```

State variables written after the call(s)

```
223              withdrawRound[msg.sender] = 0;
```

External call(s)

```
322              uint256 amountOut = IRouter(router).confirmAndBuy(pendingDealAmount, address(this));
```

```
324              _updateRewards();
```

- This function call executes the following external call(s).
- In `BatchRouter._updateRewards` ,

- `esNeuAmount = IRewardTracker(stakedNeuGlpTracker).claimForAccount(address(this),address(this))`

- In `BatchRouter._updateRewards`,
  - `wantAmount = IRewardTracker(feeNeuGlpTracker).claimForAccount(address(this),address(this))`

State variables written after the call(s)

```
334             pendingDealAmount = 0;
```

```
331             totalSnGlpReceivedAmount += amountOut;
```

External call(s)

```
322             uint256 amountOut =
IRouter(router).confirmAndBuy(pendingDealAmount, address(this));
```

```
324             _updateRewards();
```

- This function call executes the following external call(s).
- In `BatchRouter._updateRewards`,
  - `esNeuAmount = IRewardTracker(stakedNeuGlpTracker).claimForAccount(address(this),address(this))`

- In `BatchRouter._updateRewards`,
  - `wantAmount = IRewardTracker(feeNeuGlpTracker).claimForAccount(address(this),address(this))`

```
337             uint256 amountOut =
IRouter(router).confirmAndSell(pendingDealAmount, address(this));
```

State variables written after the call(s)

```
345         executed = false;
```

External call(s)

```
337             uint256 amountOut =
IRouter(router).confirmAndSell(pendingDealAmount, address(this));
```

State variables written after the call(s)

```
342             pendingDealAmount = 0;
```

External call(s)

```
149         IStrategyVault(strategyVault).harvest();
```

```
151         _checkLastFundingTime();
```

- This function call executes the following external call(s).
- In `RouterV2._checkLastFundingTime` ,
  - `IVault(gmxVault).updateCumulativeFundingRate(want)`

State variables written after the call(s)

```
152         _setPendingStatus(true);
```

- This function call executes the following assignment(s).
- In `RouterV2._setPendingStatus` ,
  - `pendingStatus.isProgress = true`
- In `RouterV2._setPendingStatus` ,
  - `pendingStatus.recipient = msg.sender`
- In `RouterV2._setPendingStatus` ,
  - `pendingStatus.totalValueBefore = IStrategyVault(strategyVault).totalValue()`
- In `RouterV2._setPendingStatus` ,
  - `pendingStatus.totalSupplyBefore = IERC20(nGlp).totalSupply()`
- In `RouterV2._setPendingStatus` ,

- `pendingStatus.longValueBefore = _gmxHelper.getLongValue(IERC20(fsGlp).balanceOf(strategyVault))`

- In `RouterV2._setPendingStatus`,
  - `pendingStatus.wbtcCollateralBefore = wbtcCollateral`
- In `RouterV2._setPendingStatus`,
  - `pendingStatus.wethCollateralBefore = wethCollateral`

External call(s)

```
192         IStrategyVault(strategyVault).harvest();
```

```
194         _checkLastFundingTime();
```

- This function call executes the following external call(s).
- In `RouterV2._checkLastFundingTime`,
  - `IVault(gmxVault).updateCumulativeFundingRate(want)`

State variables written after the call(s)

```
195         _setPendingStatus(true);
```

- This function call executes the following assignment(s).
- In `RouterV2._setPendingStatus`,
  - `pendingStatus.isProgress = true`
- In `RouterV2._setPendingStatus`,
  - `pendingStatus.recipient = msg.sender`
- In `RouterV2._setPendingStatus`,
  - `pendingStatus.totalValueBefore = IStrategyVault(strategyVault).totalValue()`
- In `RouterV2._setPendingStatus`,
  - `pendingStatus.totalSupplyBefore = IERC20(nGlp).totalSupply()`

- In `RouterV2._setPendingStatus`,
  - `pendingStatus.longValueBefore = _gmxHelper.getLongValue(IERC20(fsGlp).balanceOf(strategyVault))`
- In `RouterV2._setPendingStatus`,
  - `pendingStatus.wbtcCollateralBefore = wbtcCollateral`
- In `RouterV2._setPendingStatus`,
  - `pendingStatus.wethCollateralBefore = wethCollateral`

External call(s)

```
247         IStrategyVault(strategyVault).harvest();
```

```
249         _checkLastFundingTime();
```

- This function call executes the following external call(s).
- In `RouterV2._checkLastFundingTime`,
  - `IVault(gmxVault).updateCumulativeFundingRate(want)`

State variables written after the call(s)

```
250         _setPendingStatus(false);
```

- This function call executes the following assignment(s).
- In `RouterV2._setPendingStatus`,
  - `pendingStatus.isProgress = true`
- In `RouterV2._setPendingStatus`,
  - `pendingStatus.recipient = msg.sender`
- In `RouterV2._setPendingStatus`,
  - `pendingStatus.totalValueBefore = IStrategyVault(strategyVault).totalValue()`
- In `RouterV2._setPendingStatus`,

- `pendingStatus.totalSupplyBefore = IERC20(nGlp).totalSupply()`

- In `RouterV2._setPendingStatus`,
  - `pendingStatus.longValueBefore = _gmxHelper.getLongValue(IERC20(fsGlp).balanceOf(strategyVault))`

- In `RouterV2._setPendingStatus`,
  - `pendingStatus.wbtcCollateralBefore = wbtcCollateral`

- In `RouterV2._setPendingStatus`,
  - `pendingStatus.wethCollateralBefore = wethCollateral`

External call(s)

```
247         IStrategyVault(strategyVault).harvest();
```

```
249         _checkLastFundingTime();
```

- This function call executes the following external call(s).
- In `RouterV2._checkLastFundingTime`,
  - `IVault(gmxVault).updateCumulativeFundingRate(want)`

```
253             IRewardTracker(snGlp).unstakeForAccount(msg.sender, fnGlp, _amount,
msg.sender);
```

```
254             IRewardTracker(fnGlp).unstakeForAccount(msg.sender, nGlp, _amount,
address(this));
```

```
270             _validateMaxGlpAmountIn(wbtcCollateralDelta + wethCollateralDelta);
```

- This function call executes the following external call(s).
- In `RouterV2._validateMaxGlpAmountIn`,
  - `amount = IStrategyVault(strategyVault).usdToTokenMax(want,_collateralDelta,true)`

```
271             IStakedGlp(stakedGlp).transferFrom(strategyVault, address(this),
unstakeGlpAmount);
```

State variables written after the call(s)

```
272             pendingStatus.withdrawGlp = true;
```

External call(s)

```
319             _mintNeutraGlp(_requestKey);
```

- This function call executes the following external call(s).
- In `RouterV2._mintNeutraGlp`,
  - `IMintable(nGlp).mint(address(this),mintAmount)`
- In `RouterV2._mintNeutraGlp`,
  - `IMintable(nGlp).mint(address(this),mintAmount)`
- In `RouterV2._mintNeutraGlp`,
  - `IMintable(nGlp).mint(address(this),mintAmount)`

```
321             _burnNeutraGlp();
```

- This function call executes the following external call(s).
- In `RouterV2._burnNeutraGlp`,
  - `IMintable(nGlp).burn(address(this),burnAmount - cumulativeBurnAmount)`

State variables written after the call(s)

```
324         pendingStatus.fisrtCallbackExecuted = true;
```

External call(s)

```
341         _vault.confirmCallback();
```

```
346             _mintNeutraGlp(_requestKey);
```

- This function call executes the following external call(s).
- In `RouterV2._mintNeutraGlp`,
    - `IMintable(nGlp).mint(address(this),mintAmount)`
- In `RouterV2._mintNeutraGlp`,
    - `IMintable(nGlp).mint(address(this),mintAmount)`
- In `RouterV2._mintNeutraGlp`,
    - `IMintable(nGlp).mint(address(this),mintAmount)`

```
348             IRewardTracker(fnGlp).stakeForAccount(address(this), recipient,
nGlp, cumulativeMintAmount);
```

```
349             IRewardTracker(snGlp).stakeForAccount(recipient, recipient, fnGlp,
cumulativeMintAmount);
```

State variables written after the call(s)

```
351             cumulativeMintAmount = 0;
```

External call(s)

```
341         _vault.confirmCallback();
```

```
354             IMintable(nGlp).burn(address(this), remainingAmount);
```

```
359                 IRewardRouter(glpRewardRouter).mintAndStakeGlp(want, balance,
0, 0);
```

```
360                 IStakedGlp(stakedGlp).transfer(recipient,
IERC20(fsGlp).balanceOf(address(this)));
```

State variables written after the call(s)

```
361                 pendingStatus.withdrawGlp = false;
```

External call(s)

```
481         IMintable(nGlp).burn(address(this), burnAmount - cumulativeBurnAmount);
```

State variables written after the call(s)

```
483         cumulativeBurnAmount = burnAmount;
```

External call(s)

```
197         _harvest();
```

- This function call executes the following external call(s).
- In `StrategyVault._harvest`,
  - `IRewardRouter(rewardRouter).handleRewards(true,true,true,true,true,true,false)`
- In `StrategyVault._harvest`,
  - `IRouter(gmxRouter).swap(path,wethBalance,0,address(this))`
- In `StrategyVault._collectManagementFee`,
  - `IMintable(nGlp).mint(management,alpha)`

State variables written after the call(s)

```
226                     pendingPositionFeeInfo.wbtcFundingFee = fundingFee;
```

```
229                     pendingPositionFeeInfo.wethFundingFee = fundingFee;
```

```
249         _requireConfirm();
```

- This function call executes the following assignment(s).
- In `StrategyVault._requireConfirm`,

- `confirmed = false`

External call(s)

```
264         _harvest();
```

- This function call executes the following external call(s).
- In `StrategyVault._harvest`,
  - `IRewardRouter(rewardRouter).handleRewards(true,true,true,true,true,true,false)`
- In `StrategyVault._harvest`,
  - `IRouter(gmxRouter).swap(path,wethBalance,0,address(this))`
- In `StrategyVault._collectManagementFee`,
  - `IMintable(nGlp).mint(management,alpha)`

State variables written after the call(s)

```
294                     pendingPositionFeeInfo.wbtcFundingFee = fundingFee;
```

```
297                     pendingPositionFeeInfo.wethFundingFee = fundingFee;
```

```
317         _requireConfirm();
```

- This function call executes the following assignment(s).
- In `StrategyVault._requireConfirm`,
  - `confirmed = false`

External call(s)

```
456         IMintable(nGlp).mint(management, alpha);
```

State variables written after the call(s)

```
457         lastCollect = block.timestamp;
```

External call(s)

```
637         IERC20(want).approve(glpManager, 0);
```

```
638         IERC20(want).approve(gmxRouter, 0);
```

State variables written after the call(s)

```
639         want = _want;
```

External call(s)

```
706         _harvest();
```

- This function call executes the following external call(s).
- In `StrategyVault._harvest`,
  - `IRewardRouter(rewardRouter).handleRewards(true,true,true,true,true,true,false)`
- In `StrategyVault._harvest`,
  - `IRouter(gmxRouter).swap(path,wethBalance,0,address(this))`
- In `StrategyVault._collectManagementFee`,
  - `IMintable(nGlp).mint(management,alpha)`

State variables written after the call(s)

```
709             feeReserves = 0;
```

```
716         feeReserves = 0;
```

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy

attack.

Alleviation

`[CertiK]` : The team updated the code in commit ef9633222a5955081625f21f1d9aa8ed7b4a417d and partially resolved this issue.

`[Neutra]` : We will no longer use `BatchRouter` contract. Users can only interact with `instantDeposit` , `instantDepositGlp` , and `instantWithdraw` functions.

NFB-09 | STATUS IS NOT RESET TO `PositionExecutionStatus.NONE`

Category	Severity	Location	Status
Logical Issue	Medium	contracts/ExecutionCallbackTarget.sol (Neutra): 41~49; contracts/RouterV2.sol (Neutra): 315~337, 573	Resolved

Description

The function `handleFailure` is used to handle the failure once the create position fails. However, it does not reset the position execution status to `PositionExecutionStatus.NONE`.

```
41      function _successCallback(bool _isIncrease, bytes32 _requestKey) internal {
42          if (status == PositionExecutionStatus.NONE) {
43              status = PositionExecutionStatus.PARTIAL;
44              IRouter(router).firstCallback(_isIncrease, _requestKey);
45          } else if (status == PositionExecutionStatus.PARTIAL) {
46              status = PositionExecutionStatus.NONE;
47              IRouter(router).secondCallback(_isIncrease, _requestKey);
48          }
49      }
```

According to the logic above, the `PositionExecutionStatus` controls the order in which callback functions are run. If the second asset of a position is executed failure by the keeper of the GMX and the position execution status is not reset to `PositionExecutionStatus.NONE`, in the next deposit or withdrawal, the second callback function `secondCallback` will be first called. This could potentially cause the user to not receive the `snGlp` share when depositing or to not be able to withdraw the `want` asset.

Recommendation

We recommend reviewing the logic again and resetting the position execution status to `PositionExecutionStatus.NONE` when handling failure in the function `handleFailure`.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the commit ef9633222a5955081625f21f1d9aa8ed7b4a417d.

NFB-10 | POTENTIAL DAMAGE DUE TO UNPROTECTED INITIALIZER

Category	Severity	Location	Status
Logical Issue	Medium	contracts/BatchRouter.sol (Neutra): 108; contracts/StrategyVault.sol (Neutra): 148	Resolved

Description

One or more logic contracts do not protect their initializers. An attacker can call the initializer and assume ownership of the logic contract, whereby she can perform privileged operations that either indirectly break the proxy by destroying the logic contract or trick unsuspecting users into believing that she is the owner of the upgradeable contract.

Recommendation

We advise calling `_disableInitializers` in the constructor or giving the constructor the `initializer` modifier to prevent the initializer from being called on the logic contract.

Reference: https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#initializing_the_implementation_contract

Alleviation

`[Neutra]`: We are utilizing Hardhat to manage the contracts, which automatically calls the initialize function sequentially upon deployment.

RVN-02 | POTENTIAL UNSUCCESSFUL ASSETS WITHDRAW

Category	Severity	Location	Status
Logical Issue	Medium	contracts/RouterV2.sol (Neutra): 242, 253~254, 259~260, 267, 277	Acknowledged

Description

Neutra Finance will divide the assets entrusted by users into three parts, which will be used respectively for (1) purchasing GLP, (2) shorting WBTC, and (3) shorting WETH. Users will receive nGlp tokens as share certificates, with the quantity being the sum of the nGlp generated from the three parts mentioned above. The nGlp will be automatically staked, and users will ultimately receive an equal number of snGlp tokens.

When users redeem their assets from Neutra Finance using snGlp, the incoming snGlp tokens should also be divided into three parts, which will be used respectively for (1) selling GLP, (2) closing WBTC positions, and (3) closing WETH positions.

However, the current logic of the `instantWithdraw()` function in the RouterV2 contract does not split the incoming snGlp tokens, and the quantity of snGlp tokens used for selling GLP, closing WBTC positions, and closing WETH positions is equal to the quantity of snGlp tokens passed in by the user.

This may lead to the redemption operation being rolled back due to insufficient nGlp tokens, causing users to be unable to retrieve their entrusted assets, or it may result in users taking out more assets than they entrusted, as the RouterV2 contract has sufficient nGlp tokens.

Recommendation

We recommend modifying the code so that users can successfully and accurately redeem their self-hosted assets.

Alleviation

`[Neutra]`: We have decided not to split the instantWithdraw function into three parts due to the potential for price changes in the GMX contracts. Since all user nGlp must be burned during the withdrawal process, it is difficult to accurately determine the exact amount of shares that need to be redeemed.

BRN-03 | CLAIMABLE `want` AND `snGlp` DEPEND ON `_dealAmount`

Category	Severity	Location	Status
Logical Issue	Minor	contracts/BatchRouter.sol (Neutra): 251, 277, 306, 319	Acknowledged

Description

The function `__claimStakedNeuGlp()` or `__claimWant()` is used to calculate the claimable `want` or `snGlp` and transfer them to the users.

```
248     uint256 totalBalance = totalSnGlpPerRound[round];
249     uint256 totalReceived = totalWantReceivedPerRound[round];
250
251     uint256 claimAmount = totalReceived * balance / totalBalance;
```

```
277     uint256 claimAmount = totalSnGlpReceivedPerRound[round] * balance /
totalWantPerRound[round];
```

```
306     function executeBatchPositions(bool _isWithdraw, bytes[] calldata _params,
uint256 _dealAmount) external payable onlyHandlerAndAbove {
307         require(msg.value >= executionFee * 2, "BatchRouter: not enough
execution Fee");
308         uint256 amountIn = _isWithdraw ?
totalSnGlpPerRound[currentWithdrawRound] : totalWantPerRound[currentDepositRound];
309         IRouter(router).executePositionsBeforeDealGlp{value: msg.value}
(amountIn, _params, _isWithdraw);
310
311         pendingDealAmount = _dealAmount;
312         lastExecutionStatus = _isWithdraw;
313
314         executed = true;
315
316         emit ExecuteBatchPositions(_isWithdraw, amountIn);
317     }
```

```
319   function confirmAndDealGlp() external onlyHandlerAndAbove {
320         require(executed, "BatchRouter: executes positions first");
321         if (!lastExecutionStatus) {
322             uint256 amountOut =
IRouter(router).confirmAndBuy(pendingDealAmount, address(this));
323
324             _updateRewards();
325
326             totalSnGlpReceivedPerRound[currentDepositRound] = amountOut;
327             ...
328         } else {
329             uint256 amountOut =
IRouter(router).confirmAndSell(pendingDealAmount, address(this));
330             totalWantReceivedPerRound[currentWithdrawRound] = amountOut;
331             ...
332         }
333         executed = false;
334     }
```

As per the claimable amount calculation formula, the value of `totalWantReceivedPerRound[round]` or `totalSnGlpReceivedPerRound[round]` has a significant impact on the claimable amount. The amount of `totalWantReceivedPerRound[round]` or `totalSnGlpReceivedPerRound[round]` is ultimately determined by the `_dealAmount` parameter of the `executeBatchPositions` function. If the `_dealAmount` is insufficient, users may risk losing money, and it is essential to determine the appropriate quantity of `_dealAmount` to ensure that users can obtain the expected claimable tokens without any losses. Before proceeding, we would like to request confirmation from our customers on the correct approach for determining the right quantity of `_dealAmount`.

Recommendation

We recommend reviewing the logic again and ensure it is as intended.

Alleviation

`[Neutra]`: We will no longer use `BatchRouter` contract.

NFB-08 | MISSING ADD LOSSES FROM SHORTS

Category	Severity	Location	Status
Mathematical Operations	Minor	contracts/GmxHelper.sol (Neutra): 65, 109; contracts/RouterV2.sol (Neutra): 382~388	Resolved

Description

The code in lines 387 and 388 is used to calculate the proportion of WBTC and WETH in the GLP pool. Based on the calculated proportion, asset allocations for purchasing GLP and shorting WBTC and WETH are determined. The proportion calculation formula divides the assets of WBTC/WETH in the pool by the total assets in the pool.

However, the calculation of these two assets is inconsistent. When calculating the total assets in the pool, the loss incurred by users shorting is taken into account, whereas when calculating the assets of WBTC/WETH, this is not considered. This results in the proportion of WBTC/WETH being calculated as smaller than the actual value.

Recommendation

Please provide more information about the design of the calculation logic in this section.

Alleviation

`[Neutra]`: This is by design. The reason why we did not consider short profits in the `getTokenAums` function was based on the results of our backtesting. We compared the values with and without short profits and found that the results were better without them. Since the `getTokenAums` function is directly related to our hedging strategy, we will continue to perform backtesting and may consider making changes if we discover better results.

NFB-11 | LACK OF REASONABLE BOUNDARY

Category	Severity	Location	Status
Logical Issue	Minor	contracts/BatchRouter.sol (Neutra): 381; contracts/RouterV2.sol (Neutra): 605, 615; contracts/StrategyVault.sol (Neutra): 627, 647	Acknowledged

Description

The variables `_executionFee` , `_fee` , and `_bps` do not have reasonable boundaries, so they can be given arbitrary values after deploying.

Recommendation

We recommend adding reasonable upper and lower boundaries to all the configuration variables.

Alleviation

`[Neutra]` : Issue acknowledged. We will fix the issue in the future, which will not be included in this audit engagement.

NFB-12 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	contracts/BatchRouter.sol (Neutra): 109, 110, 111; contracts/ExecutionCallbackTarget.sol (Neutra): 20; contracts/GmxHelper.sol (Neutra): 59, 60, 61, 62; contracts/RepayCallbackTarget.sol (Neutra): 15; contracts/RouterV2.sol (Neutra): 599, 600, 626; contracts/StrategyVault.sol (Neutra): 639, 652; contracts/test/StrategyVaultV2.sol (Neutra): 761	Resolved

Description

Addresses should be checked before assignment or external call to make sure they are not zero addresses.

Recommendation

We advise adding a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

`[CertiK]` : The team heeded the advice and partially resolved the finding in the commit ef9633222a5955081625f21f1d9aa8ed7b4a417d.

`[Neutra]` : We do not need to perform a zero-check on the constructor since the input parameters will be validated by the script that we will run. Additionally, in the `StrategyVault` contract, the `callbackTarget` parameter can be set to the zero address.

NFB-13 | UNUSED RETURN VALUE

Category	Severity	Location	Status
Volatile Code	Minor	contracts/BatchRouter.sol (Neutra): 136; contracts/RouterV2.sol (Neutra): 135, 136, 137, 274, 359; contracts/StrategyVault.sol (Neutra): 166, 167, 169, 170, 572~583, 597~609, 637, 638, 640, 641; contracts/StrategyVaultV2.sol (Neutra): 170~181, 214	Acknowledged

Description

The return value of an external call is not stored in a local or state variable.

Recommendation

We recommend checking or using the return values of all external function calls.

Alleviation

`[Neutra]`: Issue acknowledged. We won't make any changes for the current version.

NFB-14 | UNCHECKED ERC-20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	Minor	contracts/BatchRouter.sol (Neutra): 147, 168, 205, 257, 287, 288; contracts/RouterV2.sol (Neutra): 157, 197, 216, 271, 360, 363, 581, 585; contracts/StrategyVault.sol (Neutra): 526, 546, 717, 728	Acknowledged

Description

The return value of the transfer()/transferFrom() call is not checked.

Recommendation

Since some ERC-20 tokens return no values and others return a `bool` value, they should be handled with care. We advise using the OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if `false` is returned, making it compatible with all ERC-20 token implementations.

Alleviation

`[Neutra]`: Issue acknowledged. We won't make any changes for the current version.

NFB-15 | DIVIDE BEFORE MULTIPLY

Category	Severity	Location	Status
Mathematical Operations	Minor	contracts/BatchRouter.sol (Neutra): 277, 284; contracts/RouterV2.sol (Neutra): 387, 394, 404, 408	Acknowledged

Description

Performing integer division before multiplication truncates the low bits, losing the precision of the calculation.

```
277         uint256 claimAmount = totalSnGlpReceivedPerRound[round] * balance / totalWantPerRound[round];
```

```
284         uint256 esNeuClaimable = claimAmount * (cumulativeEsNeuRewardPerToken - cumulativeEsNeuRewardPerRound[round]) / PRECISION;
```

```
387         uint256 wbtcRatio = (aums[0] * MAX_BPS) / totalAum;
```

```
394         wbtcAmountIn = remainingAmount * wbtcRatio / (wbtcRatio + wethRatio);
```

```
404         uint256 sizeDelta = positionParam.size * _amount / _totalSupply;
```

```
408         uint256 usdOut = profitParam.hasProfit ? sizeDelta * profitParam.pnl / positionParam.size : 0;
```

Recommendation

We recommend applying multiplication before division to avoid loss of precision.

Alleviation

`[Neutra]`: We are aware of the precision of calculation concern, but at the same time, there is a possibility of overflow issues as some variables have 30 decimals.

NFU-05 | INCOMPATIBILITY WITH DEFLATIONARY TOKENS

Category	Severity	Location	Status
Logical Issue	Minor	contracts/BatchRouter.sol (f738900c182fef4fd1b9f23b695db0ab5c6daa6): 147, 150, 168, 171; contracts/RouterV2.sol (f738900c182fef4fd1b9f23b695db0ab5c6daa6): 143, 146, 180, 204, 359; contracts/StrategyVault.sol (f738900c182fef4fd1b9f23b695db0ab5c6daa6): 546, 547	Acknowledged

Description

When transferring deflationary ERC20 tokens, the input amount may not be equal to the received amount due to the charged transaction fee. For example, if a user sends 100 deflationary tokens (with a 10% transaction fee), only 90 tokens actually arrived to the contract. However, a failure to discount such fees may allow the same user to withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

Recommendation

We advise the client to regulate the set of tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

Alleviation

`[Neutra]` : Issue acknowledged. We will fix the issue in the future, which will not be included in this audit engagement.

RVN-03 | MISSING REFUND THE EXECUTION FEE LEFTOVER

Category	Severity	Location	Status
Logical Issue	Minor	contracts/RouterV2.sol (Neutra): 146, 189, 242, 301	Resolved

Description

According to the logic of the `RouterV2` contract code by Neutra, users are responsible for paying the execution fees in ETH for opening or closing positions in GMX. The payable functions that involve opening or closing positions, namely `instantDeposit()`, `instantDepositGlp()`, and `instantWithdraw()`, all check whether the ETH transferred by the user is sufficient to cover the execution fees. However, the conditional statement in the code is `msg.value >= execution * n`. As a result, if the amount of ETH transferred by the user is greater than the required execution fees, there will be a surplus after deducting the GMX execution fees.

In line 301 of the `instantWithdraw()` function, the excess ETH is returned to the user. However, there is no code logic to return excess ETH in the `instantDeposit()` and `instantDepositGlp()` functions. This design is unreasonable.

Recommendation

We recommend modifying the code to refund users if they overpay for the GMX opening or closing execution fees. A potential solution is to add the following code to the `instantDeposit()` and `instantDepositGlp()` functions:

```
payable(msg.sender).transfer(address(this).balance);
```

Alleviation

`[CertiK]`: Neutra team has added the code logic to refund GMX execution fee leftovers and solved this issue in commit ef9633222a5955081625f21f1d9aa8ed7b4a417d.

RVN-04 CONSTANT CUMULATIVE FUNDING RATE UPDATE INTERVAL

Category	Severity	Location	Status
Logical Issue	Minor	contracts/RouterV2.sol (Neutra): 486~491	Resolved

Description

When accessing the functions `instantDeposit()`, `instantDepositGlp()`, and `instantWithdraw()` in the Neutra `RouterV2` contract for staking or redemption operations, if the cumulative funding rate for the `GMX` contract's staked tokens, specifically DAI, has not been updated for over an hour, the `updateCumulativeFundingRate()` function in the `GMX Vault` contract will be called to update it.

In the `updateCumulativeFundingRate()` function, the cumulative funding rate for the staked token, DAI, is also checked to see if it has not been updated for a certain duration. This duration is given by the state variable `fundingInterval`, which is set to one hour in the `Vault` contract on the chain, but can be modified to be greater than one hour by calling the `setFundingRate()` function.

However, according to the logic in the `_checkLastFundingTime()` function in the Neutra `RouterV2` contract, the cumulative funding rate for the staked token, DAI, needs to be updated every hour. If the `fundingInterval` state variable is modified to be greater than one hour, the aforementioned cumulative funding rate will not be updated every hour according to the logic in `_checkLastFundingTime()`.

Recommendation

We recommend refactoring the code to avoid the erroneous updating of the cumulative funding rate. A potential solution would be to access the GMX `Vault` contract and obtain the value of the state variable `fundingInterval`, which would then be used to determine whether or not to update the cumulative funding rate.

Alleviation

[`CertiK`]: Neutra team has utilized the state variable `fundingInterval` in the GMX `Vault` contract to aid in determining whether to update the funding rate. This issue is solved in commit ef9633222a5955081625f21f1d9aa8ed7b4a417d.

SVN-01 | LACK OF STORAGE GAP IN UPGRADEABLE CONTRACT

Category	Severity	Location	Status
Logical Issue	Minor	contracts/StrategyVault.sol (Neutra): 48	Acknowledged

Description

There is no storage gap preserved in the logic contract. Any logic contract that acts as a base contract that needs to be inherited by another upgradeable child should have a reasonable size of storage gap preserved for the new state variable introduced by the future upgrades.

Recommendation

We recommend having a storage gap of a reasonable size preserved in the logic contract in case that new state variables are introduced in future upgrades. For more information, please refer to: https://docs.openzeppelin.com/contracts/3.x/upgradeable#storage_gaps.

Alleviation

`[Neutra]`: The `StrategyVault` contract has already been deployed.

SVN-02 | LACK OF VALIDATION FOR ARRAY LENGTH

Category	Severity	Location	Status
Logical Issue	Minor	contracts/StrategyVault.sol (Neutra): 188	Acknowledged

Description

There is no validation to check whether the length of the array `_selectors` is the same length as each of the arrays `_params`. In the event that any of these lengths are less than the length of `_selectors`, there is no revert of the function.

Recommendation

We recommend the client add a require statement to validate the array length.

Alleviation

`[Neutra]`: Issue acknowledged. We will fix the issue in the future, which will not be included in this audit engagement.

SVN-03 | LACK OF VALIDATION OF `confirmed`

Category	Severity	Location	Status
Logical Issue	Minor	contracts/StrategyVault.sol (Neutra): 353	Resolved

Description

There is no validation ensure that the function only can be executed if the `confirmed` is false.

Recommendation

We recommend reviewing the logic again and adding the validation.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the commit ef9633222a5955081625f21f1d9aa8ed7b4a417d.

SVN-04 | POTENTIAL PAY DUPLICATED UNPAID FUNDING FEE

Category	Severity	Location	Status
Logical Issue	Minor	contracts/StrategyVault.sol (Neutra): 686	Resolved

Description

If the function is called after the next rebalance operation(minimiseDeltaWithBuyGlp,minimiseDeltaWithSellGlp), the unpaid funding fee will be paid repeatedly. We would like to confirm with the client when the function will be executed.

Recommendation

We recommend reviewing the logic again and ensuring it is intended.

Alleviation

`[CertiK]` : The team heeded the advice and resolved the finding in the commit ef9633222a5955081625f21f1d9aa8ed7b4a417d.

SVN-05 | EOA RECEIVE `want` WHEN SELLING `glp`

Category	Severity	Location	Status
Logical Issue	Minor	contracts/StrategyVault.sol (Neutra): 215, 282	Acknowledged

Description

The `StrategyVault` contract is responsible for using the `want` asset to purchase `glp` tokens for rebalancing. However, when `glp` tokens are sold, the `recipient` address that receives the `want` asset is specified by the input parameter instead of using the original address that paid for the `want` asset. Therefore, we would like to confirm with the client whether this current implementation aligns with the original project design.

Recommendation

We recommend reviewing the logic again and ensuring there is sufficient `want` in the contract `StrategyVault` for rebalancing.

Alleviation

`[Neutra]`: Issue acknowledged. We will fix the issue in the future, which will not be included in this audit engagement.

OPTIMIZATIONS | NEUTRA - AUDIT

ID	Title	Category	Severity	Status
NFB-16	State Variable Should Be Declared Constant	Gas Optimization	Optimization	Acknowledged
NFB-17	Variables That Could Be Declared As Immutable	Gas Optimization	Optimization	Resolved

NFB-16 | STATE VARIABLE SHOULD BE DECLARED CONSTANT

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/BatchRouter.sol (Neutra): 24, 35, 36; contracts/ExecutionCallbackTarget.sol (Neutra): 12; contracts/GmxHelper.sol (Neutra): 23, 39; contracts/StrategyVault.sol (Neutra): 56, 102	Acknowledged

Description

State variables that never change should be declared as `constant` to save gas.

Recommendation

We recommend adding the `constant` attribute to state variables that never change.

Alleviation

[`Neutra`]: Issue Acknowledged. Given that the StrategyVault.sol contract has already been deployed, it would be risky to declare variables as constant within the contract.

NFB-17 | VARIABLES THAT COULD BE DECLARED AS IMMUTABLE

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/ExecutionCallbackTarget.sol (Neutra): 13; contracts/GmxHelper.sol (Neutra): 26, 27, 28, 31, 32, 33, 35, 37, 38; contracts/RepayCallbackTarget.sol (Neutra): 10; contracts/RouterV2.sol (Neutra): 63, 64, 65, 66, 67, 69, 72, 73	Resolved

Description

The linked variables assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

We recommend declaring these variables as immutable.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the commit ef9633222a5955081625f21f1d9aa8ed7b4a417d.

APPENDIX | NEUTRA - AUDIT

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Mathematical Operations	Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

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

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

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

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

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.