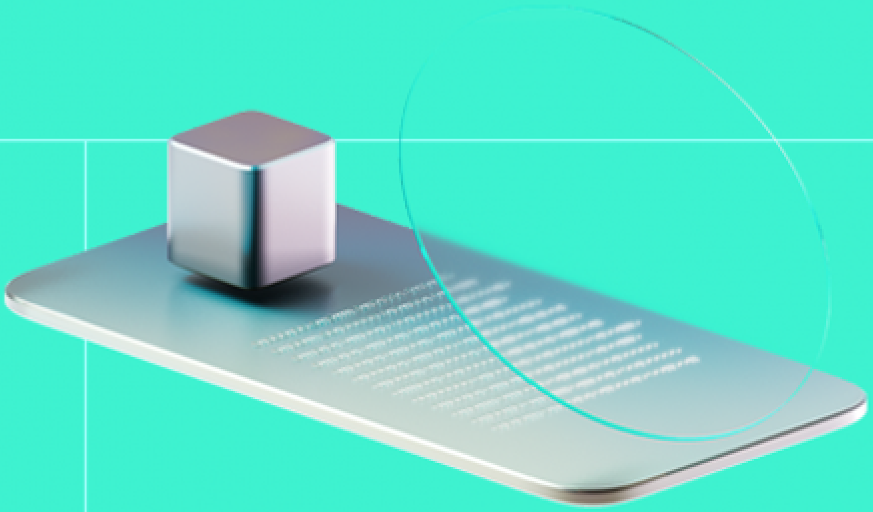
Smart Contract Code Review And Security Analysis Report

Customer: NOYA

Date: 19/04/2024



We express our gratitude to the NOYA team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

Noya is a DeFi project consisting on different yield-generating vaults that execute strategies on third party DeFi protocols.

Platform: EVM

Language: Solidity

Tags: Vault, Crosschain, DEX, EIP712, Oracle

Timeline: 26/02/2024 - 29/03/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/Noya-ai/noya-vault-contracts
Commit	95fcb52

Audit Summary

10/10	10/10	98%	9/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 9.8/10

The system users should acknowledge all the risks summed up in the risks section of the report

27	21	0	6
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	2
Medium	15
Low	10

Vulnerability	Status
F-2024-1764 - Unfinished Watchers contract allows to bypass the verification process	Mitigated
F-2024-1803 - Missing access control in checkIfTVLHasDroped allows malicious users to burn fees	Mitigated
F-2024-1806 - Too restrictive condition may lead to loss of performance fees	Mitigated
F-2024-1814 - Users can receive less tokens than entitled during withdraw	Mitigated
F-2024-1833 - Update holding position can be bypassed during bridge operation	Mitigated
F-2024-1834 - Missing slippage allows front-run in token swaps	Mitigated
F-2024-1538 - onERC721Received callback is never called which leads to loss of liquidity	Fixed
F-2024-1709 - The Keepers contract is not EIP712 compliant	Fixed
F-2024-1713 - The usage of the precompile ecrecover can lead to signature mailability	Fixed
F-2024-1716 - Missing access control in public functions leads to an unauthorized access	Fixed
F-2024-1760 - Direct reset of the minimumHealthFactor allows to bypass the MINIMUM_HEALTH_FACTOR check	Fixed
F-2024-1771 - The flashLoan process is not complete, since the assets are not transferred back in the receiveFlashLoan() callback	Fixed
F-2024-1789 - Public ERC4626 functions bypass the protocol intended ABI	Fixed
F-2024-1795 - Deposit limit check counts deposited amount twice	Fixed
F-2024-1802 - Minting of performance fees may cause temporary shares inflation	Fixed
F-2024-1808 - Missing access control in burning allow shares supply manipulation	Fixed
F-2024-1835 - Unlimited allowance is set by default	Fixed
F-2024-1836 - Use of transfer instead of call to send native assets	Fixed
F-2024-1837 - Missing call to update holding position results in miscalculated TVL	Fixed
F-2024-1839 - The non-updated totalDepositedAmount causes the incorrect calculation of the profit	Fixed
F-2024-1843 - Missing checks for the zero address	Fixed
F-2024-1846 - USD decimals are incorrectly set	Fixed
F-2024-1846 - The non-updated totalProfitCalculated causes leads to the incorrect number of fee shares	Fixed
F-2024-1847 - Chainlink’s latestRoundData might return stale or incorrect results	Fixed
F-2024-1851 - Missing return value check for tokens transfers may lead to unexpected behavior	Fixed
F-2024-1857 - Uncontrolled loop of storage interactions may lead to Denial Of Service	Fixed
F-2024-1860 - Missing revert statement causes the funds to be stuck in AccountingManager if the connector is not enabled	Fixed

Document

Name	Smart Contract Code Review and Security Analysis Report for NOYA
Audited By	David Camps Novi, Viktor Lavrenenko
Approved By	Przemyslaw Swiatowiec
Website	https://noya.ai/
Changelog	29/03/2024 - Preliminary Report ; 19/04/2024 - Final Report

Table of Contents

System Overview

Noya is a yield-generating DeFi protocol based on vaults that execute different strategies.

The project consists of several parts:

- AccountingManager: entry-point for users to deposit and withdraw tokens in exchange for shares, generating yield.
- NoyaFeeReceiver: contracts receiving the protocol fees to be managed.
- Registry.sol: centralized accountability of all vaults and positions in other protocols.
- Keepers.sol: multi-signature wallet for the execution of the vault strategies.
- NoyaGovernanceBase: defines the access control of the different roles.
- TimeLock.sol: time-lock control contract.
- Watchers: verifier contract.
- LzHelper*: contracts that interact Layer Zero messaging.
- Omnichain*: contracts to manage bridging of assets among different protocol chains.
- SwapAndBridgeHandler, LifiImplementation: contract that manage swaping and bridging of assets.
- ChainlinkOracleConnector: contract to fetch asset prices from the Chainlink protocol.
- UniswapValueOracle: TWAP oracle to get asset prices from the Uniswap protocol.
- NoyaValueOracle: defines and decides which oracle should be used to retrieve prices of assets.
- TVLHelper: contains the functions to calculate the TVL.
- Connectors: individual contracts that interact with a specific DeFi protocol (e.g. Aave, Uniswap, Frax).

Privileged roles

- Governor: responsible for changing the addresses of others.
- Maintainer: in charge of adding a new vault, adding trusted tokens for that vault, and adding trusted positions to the registry.
- Keepers: manages execution of the strategies. It's a multisig contract. Strategy managers can submit their transactions into IPFS in an encrypted way and the keepers will decrypt and execute it.
- Watchers: responsible to make sure the execution of noya is going on correctly. If there is any misbehaving (like price manipulation or any suspicious actions from the keepers) the watchers can undo the action.
- Emergency: cold wallet that is going to be used in situations that a position is stuck or another role of noya is compromised.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation Quality score is **9** out of **10**.

- Functional requirements are sufficient.
- Technical description is limited.
  - NatSpec is not sufficient.
  - Run instructions are provided.
  - Technical description is included.

Code quality

The total Code Quality score is **10** out of **10**.

- The gas modeling is correct.
- Best practices are followed.
- The development environment is configured.

Test coverage

Code coverage of the project is **98%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- All contracts are tested.
- Negative test coverage is provided.

Security score

Upon auditing, the code was found to contain **0** critical, **2** high, **15** medium, and **10** low severity issues, leading to a security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **9.8**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- The project utilizes Solidity version 0.8.20 or higher, which includes the introduction of the PUSH0 (0x5f) opcode. This opcode is currently supported on the Ethereum mainnet but may not be universally supported across other blockchain networks. Consequently, deploying the contract on chains other than the Ethereum mainnet, such as certain Layer 2 (L2) chains or alternative networks, might lead to compatibility issues or execution errors due to the lack of support for the PUSH0 opcode. In scenarios where deployment on various chains is anticipated, selecting an appropriate Ethereum Virtual Machine (EVM) version that is widely supported across these networks is crucial to avoid potential operational disruptions or deployment failures.
- The usage of a custom `minimumHealthFactor` in the following connectors: `AaveConnector.sol`, `CompoundConnector.sol`, `PrismaConnector.sol`, `SiloConnector.sol`, `FraxConnector.sol`, `MorphoBlueConnector.sol` creates a risk that the custom `minimumHealthFactor` will be improperly set by the `onlyMaintainer`, which will lead to issues.
- The value of the fees implemented by the system can change after a user makes a deposit. Thus it is possible a user will interact with the protocol expecting a specific fee (e.g. 5% withdrawal fee) but pay a higher fee later on (e.g. withdrawal fee increased to 10%).
- When performing a deposit into the protocol, some ERC20 tokens `shares` are minted to the specified `receiver` address. It should be noted that this address should be able to manage those tokens (e.g. EOA). Else, EOA should not be allowed by the system.
- The fees are implemented over the same amounts in a redundant matter: management fees, performance fees and withdrawal fees are applied over the same amounts. Although they have different principles applied, it should be noted that they are redundant.
- Any actor can send tokens to the `AccountingManager` contract in order to increase the TVL, affecting the amount of fees recorded, as well as the amount of shares minted by the user during deposits and the amount of received tokens during `withdraw`. This is due to the fact that `TVL` depends on `balanceOf` the contract.
- Both `AccountingManager` and `LifiImplementation` include functions that allow the retrieval of all native or ERC20 tokens from the contract. It should be noted, however, that those functions have restrictive access control mechanisms.
- The project is highly centralized, since it fully controls the timings of the deposits and withdrawals of the clients. This can result in Denial of Service if the protocol owner is not able to access its account to execute the required methods.
- Many of the processes and data used in the protocol are off-chain and thus cannot be audited. One example is the usage of the multi-signature process in the `Keepers` contract. Another example is the usage of `data` and `additionalData` in the holding positions structs. One additional example is the usage of cross-chain communication via `Layer Zero` and `Omnichain`.
- There is no on-chain method that checks that `BridgeRequests` are not replayed. Therefore there is a risk that the same bridging action is used repeatedly.
- The protocol uses TWAP oracles to retrieve the price of assets. It should be noted that TWAP oracles require a uniform liquidity distribution across the fetched pool in order to provide consistent and precise values. The development team should be very careful that this is the case when retrieving data from such oracles, and monitor the used pools to make sure the quality of the pool data provided is good enough.
- It should be noted that TWAP oracles provide “smoothened” data, since it uses a time window of price data to prevent flash loan attacks or other kinds of price manipulations that are typical for AMM pools. Therefore, the selected parameters in the TWAP oracle should be chosen wisely to get the best data possible: a very wide window will be more robust to price manipulation but will not reflect the real-time valuation of assets. Due to this, it is recommended to use TWAP data only in case Chainlink data feeds are not available and, moreover, to only use TWAP for healthy pools and tokens.
- Noya interacts with several third-party contracts (such as Aave, Stargate or PancakeSwap) that are not part of this audit scope. Although the consistency within those contracts and with the calls to such external protocols has been reviewed and reported if necessary, it is assumed by the audit team that those third-party protocols are done correctly by the development team (see risk statement below).
- The performance fee will only work when the vault underlying tokens valuation increases. The reason is that, when calling `collectPerformanceFees`, the previous value of `storedProfitForFee` is assigned to `totalProfitCalculated` and, therefore, only increasing `storedProfitForFee` will pass the check.

Risk Statement

This audit report focuses exclusively on the security assessment of the contracts within the specified review scope. Interactions with out-of-scope contracts are presumed to be correct and are not examined in this audit. We want to highlight that Interactions with contracts outside the specified scope, such as:

```
/contracts/connectors/AaveConnector.sol → pool
/contracts/connectors/StargateConnector.sol → stargateRouter
```

```
/contracts/connectors/PancakeswapConnector.sol → masterchef
```

have not been verified or assessed as part of this report. This situation is present in all `connector` contracts that interact with external protocols.

While we have diligently identified and mitigated potential security risks within the defined scope, it is important to note that our assessment is confined to the isolated contracts within this scope. The overall security of the entire system, including external contracts and integrations beyond our audit scope, cannot be guaranteed.

Users and stakeholders are urged to exercise caution when assessing the security of the broader ecosystem and interactions with external contracts. For a comprehensive evaluation of the entire system, additional audits and assessments outside the scope of this report are necessary.

This report serves as a snapshot of the security status of the audited contracts within the specified scope at the time of the audit. We strongly recommend ongoing security evaluations and continuous monitoring to maintain and enhance the overall system's security.

Findings

Vulnerability Details

F-2024-1538 - onERC721Received callback is never called which leads to loss of liquidity - High

Description:

The function `PancakeswapConnector::sendPositionToMasterChef()` enables the manager to transfer the NFT LP tokens from `PancakeswapConnector` to the `MasterChefV3`. This function can be seen below.

```
function sendPositionToMasterChef(uint256 tokenId) external onlyManager { IERC721
(address(positionManager)).transferFrom(address(this), address(masterchef), token
Id);
}
```

The transfer of a NFT Liquidity position is supposed to trigger the callback function `MasterChefV3::onERC721Received()` to record all of the user's liquidity data to the state. This function can be found in the code snippet below.

```
/// @notice Upon receiving a ERC721
function onERC721Received(
address,
address _from,
uint256 _tokenId,
bytes calldata
) external nonReentrant returns (bytes4) {
if (msg.sender != address(nonfungiblePositionManager)) revert NotPancakeNFT();
DepositCache memory cache;
(
,
,
cache.token0,
cache.token1,
cache.fee,
cache.tickLower,
cache.tickUpper,
cache.liquidity,
,
,
,

) = nonfungiblePositionManager.positions(_tokenId);
if (cache.liquidity == 0) revert NoLiquidity();
uint256 pid = v3PoolPid[cache.token0][cache.token1][cache.fee];
if (pid == 0) revert InvalidNFT();
PoolInfo memory pool = poolInfo[pid];
ILMPool LMPool = ILMPool(pool.v3Pool.lmPool());
if (address(LMPool) == address(0)) revert NoLMPool();

UserPositionInfo storage positionInfo = userPositionInfos[_tokenId];

positionInfo.tickLower = cache.tickLower;
positionInfo.tickUpper = cache.tickUpper;
positionInfo.user = _from;
positionInfo.pid = pid;
// Need to update LMPool.
LMPool.accumulateReward(uint32(block.timestamp));
updateLiquidityOperation(positionInfo, _tokenId, 0);

// Update Enumerable
addToken(_from, _tokenId);
emit Deposit(_from, pid, _tokenId, cache.liquidity, cache.tickLower, cache.tickUp
per);

return this.onERC721Received.selector;
}
```

However, it doesn't happen due to the usage of `transferFrom()` which doesn't call the `onERC721Received()` callback at the end of the transaction. As a result, all of the deposited liquidity is not recorded and cannot be withdrawn later on. The Proof Of Concept test was created to demonstrate the previously mentioned issue.

Assets:

- contracts/connectors/PancakeswapConnector.sol

Status: Fixed

Classification

Severity: High

Impact:

Likelihood [1-5]: 5

Impact [1-5]: 5

Exploitability [1,2]: 1

Complexity [0-2]: 1

Final Score: 3.51 [High]

Recommendations

Remediation: Consider replacing the `transferFrom()` with `safeTransferFrom()`, inside of the `sendPositionToMasterChef()` function which will trigger the `MasterChefV3::onERC721Received().`

Remediation (revised commit: 3a95ba3): The `transferFrom()` method was replaced with `safeTransferFrom()` inside of the `PancakeswapConnector::sendPositionToMasterChef()`.

Evidences

Proof Of Concept

Reproduce:

1. Alice, who is the holder of the NFT Liquidity position on the PancakeSwapV3, transfers her position to the PancakeSwapV3Connector.
2. Then Manager calls `PancakeSwapV3Connector::sendPositionToMasterChef()` to transfer the deposited Alice's LP position to the `MasterChefV3.`
3. The position is lost, since the `MasterChefV3::onERC721Received()` is not called and liquidity is not recorded.
4. The position NFT LP cannot be withdrawn after that since it hasn't been recorded previously

Results:

```
function testTransferNFTLPToMasterChef() external {
// Get the address which has a nft lp for the usdc/dai position on PancakeSwapV3
on Binance Smart Chain
address Alice = 0x310C945683a231480fF8E9b6ecDeEcA0Fc8060C3;
vm.startPrank(Alice);
IERC721(NonfungiblePositionManager).transferFrom(Alice, address(connector), 80846
); // Alice is transferring her ERC721 position with tokenId=80846 to the connect
or
vm.stopPrank();

// Manager uses PancakeSwapV3Connector::sendPositionToMasterChef() to transfer th
e previously transferred nft position to the MasterChefV3
vm.startPrank(owner);
connector.sendPositionToMasterChef(80846);

(uint128 liquidity, uint128 boostLiquidity, int24 tickLower, int24 tickUpper, uin
t256 rewardGrowthInside, uint256 reward, address user, uint256 pid, uint256 boost
Multiplier) = masterChef.userPositionInfos(80846);
console.log("Position user: %s", user);
console.log("Liquidity of the position %s", liquidity);
vm.expectRevert(abi.encodeWithSelector(NotOwner.selector));
connector.withdraw(80846);
vm.stopPrank();
}
```

As a result, we can see that the deposited liquidity to the MasterChef is zero as well as the owner of the LP position.

```
Position user: 0x0000000000000000000000000000000000000000
Liquidity of the position 0
```

F-2024-1839 - The non-updated totalDepositedAmount causes the incorrect calculation of the profit - High

Description: The variable `totalDepositedAmount`, which is used in the calculation of the profit, is not updated anywhere, which leads to the situation that the profit would be greater than expected. It can impact the number of fees the `managementFeeReceiver` and the `performanceFeeReceiver` receives.

```
function getProfit() public view returns (uint256) {
uint256 tvl = TVL();
if (tvl + totalWithdrawnAmount > totalDepositedAmount) {
return tvl + totalWithdrawnAmount - totalDepositedAmount;
}
return 0;
}
```

Furthermore, it affects the functionality of the `checkIfTVLHasDropped()` function, which will lead to the situation when the result of the `getProfit()` function in the if condition is greater than expected. The previously mentioned condition can be found in the code snippet below.

```
function checkIfTVLHasDroped() public {
if (getProfit() < storedProfitForFee) {
_burn(address(this), preformanceFeeSharesWaitingForDistribution);
preformanceFeeSharesWaitingForDistribution = 0;
profitStoredTime = 0;
}
}
```

Assets:

- contracts/accountingManager/AccountingManager.sol

Status: Fixed

Classification

Severity: High

Impact:

Likelihood [1-5]: 5

Impact [1-5]: 4

Exploitability [0-2]: 0

Complexity [0-2]: 1

Final Score: 4.3 (High)

Recommendations

Remediation: Consider updating the `totalDepositedAmount` variable inside of the `executeDeposit()` function.

Remediation (revised commit: 95fcb52): The Noya team provided a fix during the audit and `totalDepositedAmount` now is updated in the `executeDeposit()` function.

F-2024-1716 - Missing access control in public functions leads to an unauthorized access - Medium

Description: `FraxConnector::borrowAndSupply()`, `MaverickConnector::unstake()` and `MaverickConnector::removeLiquidityFromMaverickPool()` functions should only be called by the privileged addresses, which are `keeperContract` or `emergencyManager`, which can be accessed via the `NoyaGovernanceBase::onlyManager()`. However, the functions do not restrict the caller, allowing anyone to do a set of actions:

- Using a not-protected `FraxConnector::borrowAndSupply()`, anyone can borrow and supply assets to the Frax Finance.
- Using a not-protected `MaverickConnector::unstake()` and `MaverickConnector::removeLiquidityFromMaverickPool()` anyone can unstake Liquidity tokens from staking and remove liquidity from the maverick pool respectively.

The previously mentioned actions are not intended by the system for ordinary users, hence they should not be available for them.

Assets:

- contracts/connectors/FraxConnector.sol
- contracts/connectors/MaverickConnector.sol

Status: Fixed

Classification

Severity: Medium

Impact: Likelihood [1-5]: 5

Impact [1-5]: 2

Exploitability [0-2]: 0

Complexity [0-2]: 0

Final Score: 3.5 (Medium)

Recommendations

Remediation: It is recommended to add the necessary access control modifier `onlyManager` to the affected functions.

Remediation: (revised commit: a3790de): The `onlyManager` modifier has been added to the `FraxConnector::borrowAndSupply()`, `MaverickConnector::unstake()` and `MaverickConnector::removeLiquidityFromMaverickPool()` functions.

F-2024-1764 - Unfinished Watchers contract allows to bypass the verification process - Medium

Description:	According to the documentation, the `Watchers` contract is responsible for the verification of the liquidity movements from connectors to the `AccountingManager` contracts. However, due to the fact that the `Watchers::verifyRemoveLiquidity()` is empty, this requirement is not met, which means that the movements of liquidity can be done without verification.
Assets:	• contracts/governance/Watchers.sol • contracts/helpers/BaseConnector.sol
Status:	Mitigated

Classification

Severity:	Medium
Impact:	Likelihood [1-5]: 5 Impact [1-5]: 2 Exploitability [0-2]: 1 Complexity [0-2]: 0 Final Score: 2.7 (Medium)

Recommendations

Remediation: It is recommended to fix the mismatch between the documentation and implementation by implementing the `Watchers::verifyRemoveLiquidity()` function.

Remediation (revised commit: bee7c27): the Noya team reported that the function `verifyRemoveLiquidity` is present for future project upgrades. Additionally, the `Watcher` contract was updated in this remediation commit, inheriting the `Keepers` contract in order to provide actual functionality.

F-2024-1771 - The flashLoan process is not complete, since the assets are not transferred back in the receiveFlashLoan() callback - Medium

Description: The flashloan process usually consists of two parts:

- the transfer of borrowed assets to the borrower
- the transfer of borrowed assets along with the necessary fees back to the protocol, which offers flashloan functionality.

In the current implementation of the FlashLoan Recipient, specifically within `BalancerConnector`, the `receiveFlashLoan()` function fails to fulfill its crucial responsibility of transferring the borrowed assets back. This flaw severely disrupts the entire flashloan process, rendering it non-functional. The vulnerable code can be seen in the code snippet below:

```
function receiveFlashLoan(
IERC20[] memory tokens,
uint256[] memory amounts,
uint256[] memory feeAmounts,
bytes memory userData
) external override {
require(msg.sender == address(vault));
address destinationConnector = abi.decode(userData, (address));
if (registry.isAnActiveConnector(vaultId, destinationConnector)) {
for (uint256 i = 0; i < tokens.length; i++) {
tokens[i].transfer(destinationConnector, amounts[i]);
amounts[i] = amounts[i] + feeAmounts[i]; }}
flashLoanTokens = tokens;
flashLoanAmounts = amounts;
}
```

Assets:

- contracts/connectors/BalancerFlashLoan.sol

Status: Fixed

Classification

Severity: Medium

Impact: Likelihood [1-5]: 5

Impact [1-5]: 4

Exploitability [0-2]: 1

Complexity [0-2]: 1

Final Score: 3.2 (Medium)

Recommendations

Remediation: It is recommended to implement the transfer of borrowed assets and fees if necessary in the `receiveFlashLoan()` function.

Remediation: (revised commit: 2834f83): The missing transfer was added to the `BalancerFlashLoan::receiveFlashLoan()` to transfer the tokens back to the vault.

Evidences

PC

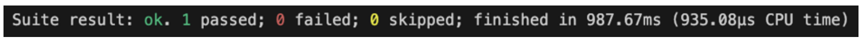
Reproduce:

1. Alice, who is the owner and the manager of the contract, tries to borrow 5e18 DAI tokens.
2. The transaction reverts, since `BalancerFlashLoan` doesn't return the borrowed assets in the `receiveFlashLoan()` callback.

```
contract TestFlashLoan is testStarter, MainnetAddresses {
UNIv3Connector uniconnector;
function setUp() public {
console.log("----------- Initialization -----------");
// -------------------------------- set env --------------------------------
uint256 fork = vm.createFork(RPC_URL, startingBlock);
vm.selectFork(fork);
console.log("Test timestamp: %s", block.timestamp);
// -------------------------------- deploy the contracts ----------------------
----------
address Alice = owner;
vm.startPrank(Alice);
deployEverythingNormal(USDC);
// -------------------------------- init connector ----------------------------
----
uniconnector = new UNIv3Connector(
uniswapV3PositionManager, uniV3Factory, BaseConnectorCP(registry, 0, swapHandler,
noyaOracle)
);
console.log("UNIv3Connector deployed: %s", address(uniconnector));
// ------------------- add connector to vaultManager -------------------
addConnectorToRegistry(vaultId, address(uniconnector));
// ------------------- add AaveConnector as eligable user for swap -------------
-----
addTrustedTokens(vaultId, address(accountingManager), USDC);
addTrustedTokens(vaultId, address(accountingManager), DAI);
addTokenToChainlinkOracle(address(USDC), address(840), address(USDC_USD_FEED));
addTokenToNoyaOracle(address(USDC), address(chainlinkOracle));
addTokenToChainlinkOracle(address(DAI), address(840), address(DAI_USD_FEED));
addTokenToNoyaOracle(address(DAI), address(chainlinkOracle));
addRoutesToNoyaOracle(address(DAI), address(USDC), address(840));
console.log("Tokens added to registry");
registry.addTrustedPosition(
vaultId, uniconnector.UNI_LP_POSITION_TYPE(), address(uniconnector), true, false,
abi.e
```

See more

Results:



```
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 987.67ms (935.08µs CPU time)
```

F-2024-1789 - Public ERC4626 functions bypass the protocol intended ABI - Medium

Description: The contract `AccountingManager` inherits the `ERC4626` standard, including all its `public` methods.

However, the protocol has defined some alternative methods that should be used by the users instead of the regular `ERC4626` ones. Since all public methods from the `ERC4626` standard are available for users to call, they can call them instead of the ones from `AccountingManager`. This way users can bypass the intended protocol entry points, resulting unexpected behaviour, loss of consistency and lack of fees.

One example of these alternative methods is `deposit`. `AccountingManager` defines `deposit` as follows:

```
function deposit(address receiver, uint256 amount, address referrer) public nonRe
entrant whenNotPaused {
if (amount == 0) {
revert NoyaAccounting_INVALID_AMOUNT();
}

baseToken.safeTransferFrom(msg.sender, address(this), amount);

if (amount > depositLimitPerTransaction) {
revert NoyaAccounting_DepositLimitPerTransactionExceeded();
}

if (TVL() + amount > depositLimitTotalAmount) {
revert NoyaAccounting_TotalDepositLimitExceeded();
}

depositQueue.queue[depositQueue.last] = DepositRequest(receiver, block.timestamp,
0, amount, 0);
emit RecordDeposit(depositQueue.last, receiver, amount, block.timestamp, referrer
);
depositQueue.last += 1;
depositQueue.totalAWFDeposit += amount;
}
```

The `ERC4626` contract's `deposit` function is:

```
function deposit(uint256 assets, address receiver) public virtual returns (uint25
6) {
uint256 maxAssets = maxDeposit(receiver);
if (assets > maxAssets) {
revert ERC4626ExceededMaxDeposit(receiver, assets, maxAssets);
}

uint256 shares = previewDeposit(assets);
_deposit(_msgSender(), receiver, assets, shares);

return shares;
}
```

Assets:

- contracts/accountingManager/AccountingManager.sol

Status: Fixed

Classification

Severity: Medium

Impact: Likelihood [1-5]: 5

Impact [1-5]: 4

Exploitability [1,2]: 1

Complexity [0-2]: 0

Final Score: 3.33 [Medium]

Recommendations

Remediation: It is recommended to block `ERC4626` `public` methods that should not be reachable by users.

Remediation (revised commit: f369e85): the following ERC4626 public methods were blocked by overriding and reverting its calls:

```
mint(uint256 shares, address receiver).
withdraw(uint256 assets, address receiver, address owner)
redeem(uint256 shares, address receiver, address shareOwner)
deposit(uint256 assets, address receiver)
```

F-2024-1802 - Minting of performance fees may cause temporary shares inflation - Medium

Description:

The performance fee process consists on several steps.

First, the fee amount, in shares, is recorded as `preformanceFeeSharesWaitingForDistribution` and minted in `recordProfitForFee`.

Later on, the function `checkIfTVLHasDroped` is called as an additional check to make sure there is no manipulation of the fee. If the check is not passed, the fees will be burned and the recorded value will be deleted:

```
function checkIfTVLHasDroped() public {
if (getProfit() < storedProfitForFee) {
_burn(address(this), preformanceFeeSharesWaitingForDistribution);
preformanceFeeSharesWaitingForDistribution = 0;
profitStoredTime = 0;
}
}
```

Finally, the fees will be collected via `collectPerformanceFees`, where the fees will be transferred to the corresponding receiver.

Due to the check in `checkIfTVLHasDroped`, there may be cases in which the fees will be burned. In those cases, there will be a temporary inflation in the amount of shares that will cause some side effects in deposits and withdrawals: since both `deposit` and `withdraw` processes depend on the `totalSupply` of `shares`, the amount of `shares` and `assets` a user will receive during this temporary inflation will be affected.

For example, a user will receive fewer assets in the withdrawal process, since the number of assets will be calculated as:

```
assets = shares_to_burn * vault_balance / total_shares
```

Assets:

- contracts/accountingManager/AccountingManager.sol

Status: Fixed

Classification

Severity: Medium

Impact:

Likelihood [1-5]: 3

Impact [1-5]: 4

Exploitability [1,2]: 1

Complexity [0-2]: 1

Final Score: 2.60 [Medium]

Recommendations

Remediation:

Consider `minting` the fees once all the checks are passed during the `performance fee` process.

Remediation (revised commit: 0d2cf98): the fees are now minted only at the latest step of the process in `collectPerformanceFees`, avoiding temporary shares inflation.

F-2024-1803 - Missing access control in checkIfTVLHasDroped allows malicious users to burn fees - Medium

Description: The method `checkIfTVLHasDroped` will burn the performance fees if the profit of the vault has dropped since it was recorded:

```
function checkIfTVLHasDroped() public {
if (getProfit() < storedProfitForFee) {
_burn(address(this), preformanceFeeSharesWaitingForDistribution);
preformanceFeeSharesWaitingForDistribution = 0;
profitStoredTime = 0;
}
}
```

Since the function is `public` and has no access control, any user can call it. Therefore, if a slightly change in the underlying token valuation would happen, `getProfit` will be lower than `storedProfitForFee` and a malicious user can burn all fees.

Assets:

- contracts/accountingManager/AccountingManager.sol

Status: Mitigated

Classification

Severity: Medium

Impact: Likelihood [1-5]: 3

Impact [1-5]: 4

Exploitability [1,2]: 1

Complexity [0-2]: 1

Final Score: 2.60 [Medium]

Recommendations

Remediation: Consider restricting the access to the function `checkIfTVLHasDroped` and/or adding some margin to the variation of `getProfit` that will burn the shares.

Remediation: the finding was mitigated given the following explanation from the Noya team:

> It is intentional because if a malicious strategy manager tries to get more fees that it's supposed to get, everyone can burn the fees if the tvl drops in that period.

F-2024-1806 - Too restrictive condition may lead to loss of performance fees - Medium

Description: The protocol performance fees are calculated from the vault underlying assets valuation via `getProfit`:

```
function recordProfitForFee() public onlyManager {
...
storedProfitForFee = getProfit();
...
}
```

Later on, a check is introduced to make sure the same valuation via `getProfit` has not decreased at all:

```
function checkIfTVLHasDroped() public {
if (getProfit() < storedProfitForFee) {
_burn(address(this), preformanceFeeSharesWaitingForDistribution);
preformanceFeeSharesWaitingForDistribution = 0;
profitStoredTime = 0;
}
}
```

Given the volatility nature of the crypto assets, rounding approximations, decimals, and other unpredictable external factors, it is very likely that this check will trigger and the fees will be lost. This is caused due to the extremely restrictive condition imposed, which does not have any margin.

Assets:

- contracts/accountingManager/AccountingManager.sol

Status: Mitigated

Classification

Severity: Medium

Impact: Likelihood [1-5]: 4

Impact [1-5]: 4

Exploitability [1,2]: 1

Complexity [0-2]: 0

Final Score: 3.03 [Medium]

Recommendations

Remediation: It is recommended to add some margin to the variation of `getProfit` so that the check is less restrictive and can support the intrinsic variations that are normal and unpredictable.

Remediation: this issue was set as `Mitigated`, provided the given explanation from Noya:

> Even if the tvl goes lower and the fees got burned, if the tvl increases again, the strategy manager will get the performance fee for both periods the next time (the fees that was burned before, will be minted again)

F-2024-1814 - Users can receive less tokens than entitled during withdraw - Medium

Description:

The function `fulfillCurrentWithdrawGroup` will set the value of `currentWithdrawGroup.totalABAmount`, which is later used to calculate the amount of tokens a user will receive during the withdraw in `executeWithdraw`.

Said variable, is set up depending on the token balance of `AccountingManager`. If the amount of tokens is enough to cover for the total tokens requested for withdraw `currentWithdrawGroup.totalCBAmount`, the process will continue as normal. However, if the balance is not enough, the users will not receive the total amount they are entitled but an amount proportional to the available assets:

```
if (availableAssets >= currentWithdrawGroup.totalCBAmount) {
currentWithdrawGroup.totalABAmount = currentWithdrawGroup.totalCBAmount;
} else {
currentWithdrawGroup.totalABAmount = availableAssets;
}
```

Later, in `executeWithdraw`, the amount the users will receive will be modified as the amount they are entitled `data.amount` multiplied by the ratio being provided by the protocol:

```
uint256 baseTokenAmount =
data.amount * currentWithdrawGroup.totalABAmount / currentWithdrawGroup.totalCBAm
ountFullfilled;
```

Furthermore, if the user withdraws all of their shares, but the `accountingManager` doesn't have the necessary number of assets, all of the user's shares are burned. The vulnerable code can be seen in the code snippet below:

```
uint256 baseTokenAmount =
data.amount * currentWithdrawGroup.totalABAmount / currentWithdrawGroup.totalCBAm
ountFullfilled;
withdrawRequestsByAddress[data.owner] -= shares;
_burn(data.owner, shares);
```

The `_burn()` function burns all shares, although the amount to be withdrawn can be different

Assets:

- contracts/accountingManager/AccountingManager.sol

Status: Mitigated

Classification

Severity: Medium

Impact:

Likelihood [1-5]: 3

Impact [1-5]: 4

Exploitability [1,2]: 1

Complexity [0-2]: 1

Final Score: 2.60 [Medium]

Recommendations

Remediation:

Users should get the tokens they are entitled to. It is recommended to either revert the call when the scenario is not possible, or to record the remaining balance of each user so that they can withdraw the tokens at a later time.

Remediation (revised commit: bee7c27): mitigated after the following feedback from the Noya team:

It is intended because the process of withdrawal, sometimes has some hidden fees and if the users that are withdrawing don't pay those fees, other users in the vault will pay and we don't want that.
So the users that are withdrawing will pay the withdrawal fee of the protocols (if there is any withdrawal fee and the bridge slippage from other chains)

F-2024-1834 - Missing slippage allows front-run in token swaps - Medium

Description:

The function `executeSwap` is responsible for calling the corresponding swap implementation and execute a token swap.

The `swapRequest` inputted into this method is the following, containing two critical parameters `checkForSlippage` and `minAmount`:

```
struct SwapRequest {
address from;
uint256 routeId;
uint256 amount;
address inputToken;
address outputToken;
bytes data;
bool checkForSlippage;
uint256 minAmount;
}
```

However, both parameters can have different values, which will be check in `executeSwap`:

```
function executeSwap(SwapRequest memory _swapRequest)
external
payable
onlyEligibleUser
onlyExistingRoute(_swapRequest.routeId)
returns (uint256 _amountOut)
{
if (_swapRequest.amount == 0) revert InvalidAmount();
RouteData memory swapImplInfo = routes[_swapRequest.routeId];
if (swapImplInfo.isBridge) revert RouteNotAllowedForThisAction();

if (_swapRequest.checkForSlippage && _swapRequest.minAmount == 0) {
uint256 _slippageTolerance = slippageTolerance[_swapRequest.inputToken][_swapRequ
est.outputToken];
if (_slippageTolerance == 0) {
_slippageTolerance = genericSlippageTolerance;
}
INoyaValueOracle _priceOracle = INoyaValueOracle(valueOracle);
uint256 _outputTokenValue =
_priceOracle.getValue(_swapRequest.inputToken, _swapRequest.outputToken, _swapReq
uest.amount);

_swapRequest.minAmount = (((1e6 - _slippageTolerance) * _outputTokenValue) / 1e6)
;
}

_amountOut = ISwapAndBridgeImplementation(swapImplInfo.route).performSwapAction(m
sg.sender, _swapRequest);

emit ExecutionCompleted(
_swapRequest.routeId, _swapRequest.amount, _amountOut, _swapRequest.inputToken, _
swapRequest.outputToken
);
}
```

In case that `checkForSlippage = false` and `minAmount = 0`, the call to `performSwapAction` will go through, without any slippage defined. As a consequence, when the swap transaction reaches the corresponding implementation, it will be executed without this critical parameter, which will allow the swap to be front-run by external actors and result in a loss of funds.

Assets:

- contracts/helpers/SwapHandler/GenericSwapAndBridgeHandler.sol

Status: Mitigated

Classification

Severity: Medium

Impact:

Likelihood [1-5]: 4

Impact [1-5]: 4

Exploitability [1,2]: 1

Complexity [0-2]: 1

Final Score: 2.91 [Medium]

Recommendations

Remediation:

It is recommended to always set a minimum amount or slippage during swaps.

Remediation (revised commit: 79c0b24): this finding was set as `Mitigated` after the following explanation from the Noya team, plus the following notes on the different entry points that are sensitive to front-running.

> When we are using this function, (the "swapHoldings" function in BaseConnector) we are actually hardcoding "true" for the checkSlippage argument, So It's going to check for slippage in all of swaps

Additionally, the function `GenericSwapAndBridgeHandler::executeSwap` is `external` and thus can be called directly, overcoming the hardcoded value mentioned by the team for the mitigation. However, the function calls are protected by the modifier `onlyEligibleUser` which will prevent random actors from executing the function. By implementing this modifier, the Noya team is responsible for the correct usage of such function in order to avoid this front-running issue.

One more entry point was found in the function `LifiImplementation::performSwapAction`, which was addressed by adding the `onlyHandler` modifier, making sure external actors cannot use the call. It should be noted that the `handler` is now responsible to make calls with the corresponding parameters in order to avoid front-running, since it is still possible.

F-2024-1835 - Unlimited allowance is set by default - Medium

Description: In `_setAllowance`, when the desired `amount` to operate is lower than the current `allowance`, an unlimited `allowance` is set:

```
function _setAllowance(IERC20 token, address spender, uint256 amount) internal {/
/audit-ok
...
if (allowance < amount) {
if (allowance != 0) {
token.approve(spender, 0);
}
token.approve(spender, type(uint256).max);
}
}
```

It is not recommended to set an `allowance` to an unlimited value, since it can result in unexpected behaviour, and allow the protocol to perform unwanted actions that are out of control.

Assets:

- contracts/helpers/SwapHandler/Implementaions/LifiImplementation.sol

Status: Fixed

Classification

Severity: Medium

Impact: Likelihood [1-5]: 3

Impact [1-5]: 4

Exploitability [1,2]: 1

Complexity [0-2]: 0

Final Score: 2.72 [Medium]

Recommendations

Remediation: Set the `allowance` to the required `amount` instead to an unlimited value.

Remediation (revised commit: bee7c27): this issue was fixed by setting the correct allowance at every time via `ERC20::forceApprove`.

F-2024-1837 - Missing call to update holding position results in miscalculated TVL - Medium

Description:

The function `addLiquidity` is called when token deposits are processed by the system. It will perform a sub-call to `_addLiquidity`. This latest function will return the bool `addPositions` in order to determine if the holding position should be updated for that token:

```
function addLiquidity(address[] memory tokens, uint256[] memory amounts, bytes me
mory data) external override {
...
bool addPositions = _addLiquidity(tokens, amounts, data); // call the specific im
plementation if the connector needs to do something after the liquidity is added
if (addPositions) {
for (uint256 i = 0; i < tokens.length; i++) {
_updateTokenInRegistry(tokens[i]); // update the token in the registry
}
}
}
```

The function `_addLiquidity` is an empty method that each `connector` should override. However, said function is not implemented in most `connectors` and, as a result, the holding positions will not be updated. In turn, the protocol `TVL` calculated via `getTVL` will not account these amounts transferred to the connectors, resulting in miscalculations. These miscalculations will affect the amount of `shares` and `tokens` obtained by the users in `deposits` and `withdraws`, as well as the calculation of `fees`.

Assets:

- contracts/helpers/BaseConnector.sol

Status: Fixed

Classification

Severity: Medium

Impact:

Likelihood [1-5]: 5

Impact [1-5]: 4

Exploitability [1,2]: 1

Complexity [0-2]: 0

Final Score: 3.33 [Medium]

Recommendations

Remediation:

It is recommended to execute `_updateTokenInRegistry` regardless of the call to `_addLiquidity`.

Remediation (revised commit: bee7c27): the call to `_updateTokenInRegistry` is now executed regardless of the `return` value of `_addLiquidity`.

F-2024-1846 - USD decimals are incorrectly set - Medium

Description: According to the Noya team, the USD token that is used as a basis is `USDC`. Said token has `6 decimals` in the principal chains where it is used. However, it is wrongly assumed it has `8 decimals`:

```
function getTokenDecimals(address token) public view returns (uint256) {
if (token == ETH) return 10 ** 18;
if (token == USD) return 10 ** 8;
uint256 decimals = IERC20Metadata(token).decimals();
return 10 ** decimals;
}
```

As a consequence, inconsistencies in calculations may arise, resulting in unexpected behavior and miscalculations.

Additionally, it is assumed that the `address(0)` is describing `ether` and `address(840)` is `describing USD`. These assumptions are dangerous since new chains and tokens may be used in the future in the protocol, that may not be compatible with the aforementioned assumptions.

In `GearBoxV3` contract, it is assumed that `address(840)` is `describing USD`.

Assets:

- contracts/helpers/valueOracle/oracles/ChainlinkOracleConnector.sol

Status: Fixed

Classification

Severity: Medium

Impact: Likelihood [1-5]: 4

Impact [1-5]: 4

Exploitability [1,2]: 1

Complexity [0-2]: 1

Final Score: 2.91 [Medium]

Recommendations

Remediation: It is recommended to work with the specific ERC20 tokens that will be actually used in the protocol. According to the Noya team this means `USDC`, `WEHT` and `WBTC`. As such, the safest approach to implement is to call the corresponding `ERC20` contract's `decimals` function in order to retrieve the right decimal number in all cases instead of hardcoding values.

Remediation: (revised commit: 27891c5): The number of decimals is not hardcoded and is calculated dynamically now.

F-2024-1846 - The non-updated totalProfitCalculated causes leads to the incorrect number of fee shares - Medium

Description: The variable `totalProfitCalculated`, which is used in the calculation of the fee shares, is not updated anywhere, which leads to the situation that the number of shares the `performanceFeeReceiver` will get, will be bigger than expected. The code snippet can be found below.

```
function recordTVLForFee() public onlyManager {
if (preformanceFeeSharesWaitingForDistribution > 0) {
_burn(address(this), preformanceFeeSharesWaitingForDistribution);
preformanceFeeSharesWaitingForDistribution = 0;
}
storedProfitForFee = getProfit();
profitStoredTime = block.timestamp;

if (storedProfitForFee < totalProfitCalculated) {
return;
}

_mint(performanceFeeReceiver, previewDeposit(((storedProfitForFee - totalProfitCa
lculated) * performanceFee) / FEE_PRECISION));
}
```

Assets:

- contracts/accountingManager/AccountingManager.sol

Status: Fixed

Classification

Severity: Medium

Impact: Likelihood [1-5]: 5

Impact [1-5]: 4

Exploitability [0-2]: 1

Complexity [0-2]: 1

Final Score: 3.2 (Medium)

Recommendations

Remediation: Consider updating the value of `totalProfitCalculated` in the code.

Remediation: (revised commit: 95fcb52): The Noya team provided a fix during the audit and `totalProfitCalculated` now is updated in the `collectPerformanceFees()` function.

F-2024-1847 - Chainlink’s latestRoundData might return stale or incorrect results - Medium

Description:

The function `getValueFromChainlinkFeed` calls chainlink's `latestRoundData`.

If there is a problem with Chainlink starting a new round and finding consensus on the new value for the oracle (e.g. Chainlink nodes abandon the oracle, chain congestion, vulnerability/attacks on the Chainlink system) consumers of this contract may continue using outdated stale or incorrect data (if oracles are unable to submit no new round is started):

```
(
/*uint80 roundID*/,
int price,
/*uint startedAt*/,
uint256 updatedAt,
/*uint80 answeredInRound*/
) = priceFeed.latestRoundData();
```

Assets:

- contracts/helpers/valueOracle/oracles/ChainlinkOracleConnector.sol

Status: Fixed

Classification

Severity: Medium

Impact:

Likelihood [1-5]: 2

Impact [1-5]: 5

Exploitability [1,2]: 1

Complexity [0-2]: 0

Final Score: 3.5 [Medium]

Recommendations

Remediation:

It is recommended to add an additional sanity check to the return values from Chainlink:

```
( roundId, price, , updatedAt, answeredInRound ) = latestRoundData();
require(price > 0, "Chainlink price <= 0");
```

Remediation (revised commit: bee7c27): the issue was fixed by implementing the following check on the received `price` in `getValueFromChainlinkFeed`

```
if (price <= 0) {
revert NoyaChainlinkOracle_PRICE_ORACLE_UNAVAILABLE(address(source), address(0),
address(0));
}
```

F-2024-1857 - Uncontrolled loop of storage interactions may lead to Denial Of Service - Medium

Description:

The functions `removeTrustedPosition` can reach the block Gas limit and become unusable, since it depends on the number of `holdingPosition` structs events recorded in each `vault`, which is unlimited.

```
function removeTrustedPosition(uint256 vaultId, bytes32 _positionId)
external
onlyVaultMaintainer(vaultId)
vaultExists(vaultId)
{
Vault storage vault = vaults[vaultId];
if (!vault.trustedPositionsBP[_positionId].isEnabled) revert NotExist();
for (uint256 i = 0; i < vault.holdingPositions.length; i++) {
if (vault.holdingPositions[i].positionId == _positionId) {
revert CannotRemovePosition(vaultId, _positionId);
}
}
}
```

Similarly, the functions `getTVL` and `getLatestUpdateTime` in `TVLHelper` also fit in this scenario.

The aforementioned array can reach an unlimited size and, as eventually, the function execution can reach the block Gas limit. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. As a consequence, the function will not be executable, resulting in a Denial of Service of `removeTrustedPosition`.

Assets:

- contracts/accountingManager/registry.sol
- contracts/helpers/TVLHelper.sol

Status: Fixed

Classification

Severity: Medium

Impact:

Likelihood [1-5]: 3

Impact [1-5]: 4

Exploitability [1,2]: 1

Complexity [0-2]: 1

Final Score: 3.3 [Medium]

Recommendations

Remediation:

Consider adding a limit to the size of `vault.holdingPositions`. Alternatively, introduce an array indexation to the arguments of reported functions, so that only a certain interval of the arrays will be processed.

Remediation (revised commit: 59dc654): this issue was mitigated by the Noya team by limiting the amount of holding positions the registry can track via `maxNumHoldingPositions`. This variable is set to `20` by default, and can be updated using `setMaxNumHoldingPositions`, which allows the upgrade of the holding positions amounts up to a hard cap of `MAX_NUM_HOLDING_POSITIONS` (`constant` state variable).

The function `updateHoldingPositions` will check that `maxNumHoldingPositions` is not surpassed.

Additional testing was performed by the Noya team to make sure the holding positions will not reach the block gas limit before the hard cap of `40` holding positions, although the tests were not provided.

F-2024-1709 - The Keepers contract is not EIP712 compliant - Low

Description:

According to the EIP-712 standard, for data structures that will be a part of the signing message, a typehash must be defined and signed. However, in the current implementation of the `Keepers` contract, the TXTYPE_HASH of the data structure is not included in the generation of the txInputHash value, which can be seen in the code snippet below. Moreover, the number of parameters used in the txInputHash is different from the number of the types in the TXTYPE_HASH.

```
function execute(
address[] memory destination,
bytes[] memory data,
uint256[] memory gasLimit,
address executor,
bytes32[] memory sigR,
bytes32[] memory sigS,
uint8[] memory sigV
) public {
require(isOwner[msg.sender]);
require(sigR.length == threshold);
require(sigR.length == sigS.length && sigR.length == sigV.length);
require(executor == msg.sender);
{
bytes32 txInputHash = keccak256(abi.encode(nonce, destination, data, gasLimit, ex
ecutor));
bytes32 totalHash = keccak256(abi.encodePacked("\x19\x01", _domainSeparatorV4(),
txInputHash));
address lastAdd = address(0);
for (uint256 i = 0; i < threshold; i++) {
address recovered = ecrecover(totalHash, sigV[i], sigR[i], sigS[i]);
require(recovered > lastAdd && isOwner[recovered]);
lastAdd = recovered;
}
nonce++;
}
for (uint256 i = 0; i < destination.length; i++) {
(bool success,) = destination[i].call{ gas: gasLimit[i] }(data[i]);
require(success, "Transaction execution reverted.");
}
}
```

As a consequence, the issue can lead to problems with the signature verification.

Assets:

- contracts/governance/Keepers.sol

Status: Fixed

Classification

Severity: Low

Impact:

Likelihood [1-5]: 4

Impact [1-5]: 3

Exploitability [1,2]: 2

Complexity [0-2]: 1

Final Score: 2.21 [Low]

Recommendations

Remediation:

It is recommended to define a proper TXTYPE_HASH for the singing data and include it in the `txInputHash` generation.

Remediation (revised commit: 774997c): The TXTYPE_HASH now contains the deadline parameter. The parameters in the `txInputHash` generation adhere to the elements in the defined TXTYPE_HASH. The TXTYPE_HASH is included in the generation of the `txInputHash`.

F-2024-1713 - The usage of the precompile ecrecover can lead to signature mailability - Low

Description: The function `execute()` is found to be vulnerable to a signature malleability issue. This vulnerability stems from the function's inability to discern between legitimately unique signatures and those that have been manipulated but are still considered valid by the Ethereum blockchain's signature verification standards. By exploiting this flaw, one of the multi-signature wallet owners can create signatures that will be accepted by the system, enabling unauthorized and repetitive transactions. The vulnerable piece of code can be found below.

```
function execute(
address[] memory destination,
bytes[] memory data,
uint256[] memory gasLimit,
address executor,
bytes32[] memory sigR,
bytes32[] memory sigS,
uint8[] memory sigV
) public {
require(isOwner[msg.sender]);
require(sigR.length == threshold);
require(sigR.length == sigS.length && sigR.length == sigV.length);
require(executor == msg.sender);
{
bytes32 txInputHash = keccak256(abi.encode(nonce, destination, data, gasLimit, ex
ecutor));
bytes32 totalHash = keccak256(abi.encodePacked("\x19\x01", _domainSeparatorV4(),
txInputHash));
address lastAdd = address(0);
for (uint256 i = 0; i < threshold; i++) {
address recovered = ecrecover(totalHash, sigV[i], sigR[i], sigS[i]);
require(recovered > lastAdd && isOwner[recovered]);
lastAdd = recovered;
}
nonce++;
}
for (uint256 i = 0; i < destination.length; i++) {
(bool success,) = destination[i].call{ gas: gasLimit[i] }(data[i]);
require(success, "Transaction execution reverted.");
}
}
```

Assets:

- contracts/governance/Keepers.sol

Status: Fixed

Classification

Severity: Low

Impact: Likelihood [1-5]: 3

Impact [1-5]: 4

Exploitability [1,2]: 2

Complexity [0-2]: 2

Final Score: 2.12 [Low]

Recommendations

Remediation: To enhance the security of your Solidity smart contracts and mitigate the risk of signature malleability attacks, it is advisable to use OpenZeppelin's ECDSA library instead of the built-in `ecrecover` function. The ECDSA library provides robust and reliable signature verification, reducing the vulnerability to replay attacks and ensuring the integrity of the contract interactions.

Remediation (revised commit: 6deba54): The EVM precompile `ecrecover()` was replaced with OpenZeppelin's `ECDSA.recover()`.

F-2024-1760 - Direct reset of the minimumHealthFactor allows to bypass the MINIMUM_HEALTH_FACTOR check - Low

Description: To update the `minimumHealthFactor` value, the following requirement `_minimumHealthFactor > MINIMUM_HEALTH_FACTOR` should be followed. However, in the `constructor()` of `SiloConnector.sol` contract, it doesn't happen. As a consequence, the necessary check is bypassed, which allows the `minimumHealthFactor` to be smaller than `MINIMUM_HEALTH_FACTOR`.

Assets:

- contracts/connectors/SiloConnector.sol

Status: Fixed

Classification

Severity: Low

Impact:

Likelihood [1-5]: 5

Impact [1-5]: 2

Exploitability [0-2]: 2

Complexity [0-2]: 0

Final Score: 2.3 (Low)

Recommendations

Remediation: It is recommended to implement the necessary condition `_minimumHealthFactor > MINIMUM_HEALTH_FACTOR` to ensure that the value of the `_minimumHealthFactor` is in the correct range.

Remediation (revised commit: bee7c27): the value of `minimumHealthFactor` was set in the `constructor` to the default `MINIMUM_HEALTH_FACTOR`.

F-2024-1795 - Deposit limit check counts deposited amount twice - Low

Description:

The `deposit` method includes a check to limit the amount of tokens deposited, which is calculated as follows:

```
if (TVL() + amount > depositLimitTotalAmount) {
revert NoyaAccounting_TotalDepositLimitExceeded();
}
```

However, the function `TVL()` included in the check already includes the deposited amount via `balanceOf(address(this))`, resulting in a duplicity of `amount`.

```
function TVL() public view returns (uint256) {
return TVLHelper.getTVL(vaultId, registry, address(baseToken)) + baseToken.balanc
eOf(address(this)) - depositQueue.totalAWFDeposit;
}
```

Assets:

- contracts/accountingManager/AccountingManager.sol

Status: Fixed

Classification

Severity: Low

Impact:

Likelihood [1-5]: 3

Impact [1-5]: 2

Exploitability [1,2]: 1

Complexity [0-2]: 0

Final Score: 2.08 [Low]

Recommendations

Remediation:

It is recommended to delete `amount` from the check, as follows:

```
if (TVL() > depositLimitTotalAmount) {
revert NoyaAccounting_TotalDepositLimitExceeded();
}
```

Remediation (revised commit: bee7c27): the excess `amount` was deleted from the check as recommended.

F-2024-1808 - Missing access control in burning allow shares supply manipulation - Low

Description:

The function `burnShares` allows any caller to burn their own shares. Although this functionality is intended to be called by the `feeReceiver` it does not have any access control mechanism:

```
function burnShares(uint256 amount) public {
_burn(msg.sender, amount);
}
```

Therefore, any user can use this method to burn their own shares. This will not cause major problems but will modify the desired behavior of the system by affecting the `deposit` and `withdraw` amounts, which depend on the total supply of shares.

Assets:

- contracts/accountingManager/AccountingManager.sol

Status: Fixed

Classification

Severity: Low

Impact:

Likelihood [1-5]: 2

Impact [1-5]: 3

Exploitability [1,2]: 1

Complexity [0-2]: 0

Final Score: 2.08 [Low]

Recommendations

Remediation:

It is recommended to introduce an access control mechanism that only allow calls from intended addresses, such as the fee receiver.

Remediation (revised commit: bee7c27): the issue was mitigated after the following explanation from the Noya team:

> Although burning the shares will effect the total supply of shares, it won't have any negative effects on the functionality of the `deposit` and `withdraw` functions. Any user might want to burn the shares in future. It doesn't make any problem in the calculation of the share price, it just increases the share price.

F-2024-1833 - Update holding position can be bypassed during bridge operation - Low

Description:

The function `executeBridge` in `SwapAndBridgeHandler` and the function `performBridgeAction` in `LifiImplementation` do not update the corresponding holding position when called directly.

When the protocol performs a bridge transaction, the regular flow is `OmnichainLogic.startBridgeTransaction` → `SwapAndBridgeHandler.executeBridge` → `LifiImplementation.performBridgeAction`. When going through this process, the holding position will be updated via `_updateTokenInRegistry`:

```
function startBridgeTransaction(BridgeRequest memory bridgeRequest) public onlyMa
nager {
...
swapHandler.executeBridge(bridgeRequest);
_updateTokenInRegistry(bridgeRequest.inputToken);
}
```

However, both `executeBridge` and `performBridgeAction` are `external` functions and, as such, they can be called directly, bypassing the update holding position check.

Assets:

- contracts/helpers/SwapHandler/GenericSwapAndBridgeHandler.sol
- contracts/helpers/SwapHandler/Implementaions/LifiImplementation.sol

Status: Mitigated

Classification

Severity: Low

Impact:

Likelihood [1-5]: 2

Impact [1-5]: 3

Exploitability [1,2]: 1

Complexity [0-2]: 0

Final Score: 2.08 [Low]

Recommendations

Remediation:

It is recommended to add a call to `_updateTokenInRegistry` in both `executeBridge` and `performBridgeAction`.

Mitigated: this issue was determined mitigated with the following explanation from Noya

> SwapAndBridgeHandler contract and LifiImplementation contract are not supposed to hold any positions. They don't belong to any specific vault and all vaults can use these contracts. So if an external contract is calling these two contracts, it's not effecting any vaults in the registry

F-2024-1836 - Use of transfer instead of call to send native assets - Low

Description: The functions `rescue` and `rescueFunds` uses the built-in `transfer` function for transferring native tokens.

The `transfer` function was commonly used in earlier versions of Solidity for its simplicity and automatic reentrancy protection. However, it was identified as potentially problematic due to its fixed gas limit of `2300`.

The usage of `transfer` function can lead to unintended function call revert when the receiving contract's `receive` or `fallback` functions require more than `2300` Gas for processing.

Assets:

- contracts/accountingManager/AccountingManager.sol
- contracts/helpers/SwapHandler/Implementaions/LifiImplementation.sol

Status: Fixed

Classification

Severity: Low

Impact: Likelihood [1-5]: 2

Impact [1-5]: 4

Exploitability [1,2]: 2

Complexity [0-2]: 0

Final Score: 2.08 [Low]

Recommendations

Remediation: It is recommended to use built-in `call` function instead of `transfer()` to transfer native assets. This method does not impose a gas limit, it provides greater flexibility and compatibility with contracts having more complex business logic upon receiving the native tokens. When working with then `call` function ensure that its execution is successful by checking the returned boolean value. It is also recommended to fallow the Check-Effects-Interactions (CEI) pattern in every case to prevent reentrancy issues.

Remediation (revised commit: bee7c27): the reported `transfer` methods were updated to `call` as recommended.

F-2024-1843 - Missing checks for the zero address - Low

Description: In Solidity, the Ethereum address `0x0000000000000000000000000000000000000000` is known as the "zero address". This address has significance because it is the default value for uninitialized address variables and is often used to represent an invalid or non-existent address. The "
Missing zero address control" issue arises when a Solidity smart contract does not properly check or prevent interactions with the zero address, leading to unintended behavior.
For instance, a contract might allow tokens to be sent to the zero address without any checks, which essentially burns those tokens as they become irretrievable. While sometimes this is intentional, without proper control or checks, accidental transfers could occur.

Assets:

- contracts/accountingManager/AccountingManager.sol
- contracts/accountingManager/NoyaFeeReceiver.sol
- contracts/accountingManager/registry.sol
- contracts/connectors/AaveConnector.sol
- contracts/connectors/AerodromeConnector.sol
- contracts/connectors/BalancerConnector.sol
- contracts/connectors/CompoundConnector.sol
- contracts/connectors/CurveConnector.sol
- contracts/connectors/FraxConnector.sol
- contracts/connectors/LidoConnector.sol
- contracts/connectors/MaverickConnector.sol
- contracts/connectors/MorphoBlueConnector.sol
- contracts/connectors/PancakeswapConnector.sol
- contracts/connectors/PendleConnector.sol
- contracts/connectors/PrismaConnector.sol
- contracts/connectors/SiloConnector.sol
- contracts/connectors/StargateConnector.sol
- contracts/governance/NoyaGovernanceBase.sol
- contracts/governance/TimeLock.sol
- contracts/helpers/LZHelpers/LZHelperReceiver.sol
- 6 more asset(s) affected

Status: Fixed

Classification

Severity: Low

Impact: Likelihood [1-5]: 2
Impact [1-5]: 2
Exploitability [1-2]: 1
Complexity [0-2]: 0
Final Score: 2.0 (Low)

Recommendations

Remediation: It is strongly recommended to implement checks to prevent the zero address from being set during the initialization of contracts. This can be achieved by adding require statements that ensure address parameters are not the zero address.

Remediation (revised commit: 6f42e71): The necessary `address(0)` checks have been added to the aforementioned assets.

F-2024-1851 - Missing return value check for tokens transfers may lead to unexpected behavior - Low

Description:	Both `transfer` and `transferFrom` ERC20 functions are called to transfer tokens. However, a `return` value check is required for these calls. Not all ERC20 tokens are guaranteed to `revert` on failure; some may `return` a boolean value (`false`) instead. If the system interacts with such tokens, a failed transfer would not cause the transaction to `revert`, potentially leading to discrepancies in the contract's state. Other ERC20 do not return anything at all when calling `transfer` and `transferFrom`. Therefore, every call will be reverted when the `return` value is checked.
Assets:	• contracts/connectors/PendleConnector.sol • contracts/connectors/BalancerFlashLoan.sol
Status:	Fixed

Classification

Severity:	Low
Impact:	Likelihood [1-5]: 3 Impact [1-5]: 4 Exploitability [1,2]: 2 Complexity [0-2]: 1 **Final Score:** 2.2 [Low]

Recommendations

Remediation:	Implement the SafeERC20 library to check the `return` value of the calls to ERC20 `transfer` and `transferFrom`, as well as interacting safely with tokens that do not return anything at all. **Remediation (revised commit: bee7c27):** the `SafeERC20` library was implemented in the reported contracts' calls to ERC20 `transfer` and `transferFrom` methods.

F-2024-1860 - Missing revert statement causes the funds to be stuck in AccountingManager if the connector is not enabled - Low

Description:

The function `AccountingManager::executeDeposit()` ensures that the deposited funds are not deposited to the inactive connector. However, when the connector is not active, the function `executeDeposit()` doesn't revert, which leads to the situation that the deposit request is processed, but the funds are still in the `AccountingManager.sol`. The function can be seen in the code snippet below.

```
function executeDeposit(uint256 maxI, address connector, bytes memory addLPdata)
public onlyManager whenNotPaused {
uint256 firstTemp = depositQueue.first;
uint64 i = 0;
uint256 processedBaseTokenAmount = 0;

while (
depositQueue.middle > firstTemp
&& depositQueue.queue[firstTemp].calculationTime + depositWaitingTime <= block.ti
mestamp && i < maxI
) {
i += 1;
DepositRequest memory data = depositQueue.queue[firstTemp];

emit ExecuteDeposit(
firstTemp, data.receiver, block.timestamp, data.shares, data.amount, data.shares
* 1e18 / data.amount
);
// minting shares for receiver address
_mint(data.receiver, data.shares);

processedBaseTokenAmount += data.amount;
delete depositQueue.queue[firstTemp];
firstTemp += 1;
}
depositQueue.totalAWFDeposit -= processedBaseTokenAmount;

totalDepositedAmount += processedBaseTokenAmount;

if (registry.isAnActiveConnector(vaultId, connector) && processedBaseTokenAmount
> 0) {
uint256[] memory amounts = new uint256[](1);
amounts[0] = processedBaseTokenAmount;
address[] memory tokens = new address[](1);
tokens[0] = address(baseToken);
IConnector(connector).addLiquidity(tokens, amounts, addLPdata);
}

depositQueue.first = firstTemp;
}
```

Assets:

- contracts/accountingManager/AccountingManager.sol

Status: Fixed

Classification

Severity: Low

Impact:

Likelihood [1-5]: 3

Impact [1-5]: 3

Exploitability [0-2]: 2

Complexity [0-2]: 1

Final Score: 2.0 (Low)

Recommendations

Remediation:

It is recommended to add the missing else scenario with a revert statement to the `AccountingManager::executeDeposit()` to ensure that the deposit request is not processed if the connector is not active.

Remediation (revised commit: bee7c27): the scenario in which the condition `registry.isAnActiveConnector(vaultId, connector) &&`

`processedBaseTokenAmount > 0)` is not fulfilled was implemented with a `revert` as recommended.

Observation Details

F-2024-1426 - Missing two-step transfer of ownership introduces risks - Info

Description: Ownable2Step prevents the contract ownership from mistakenly being transferred to an address that cannot handle it (e.g. due to a typo in the address), by requiring that the recipient of the owner permissions actively accept via a contract call of its own.

Assets:

- contracts/governance/Keepers.sol
- contracts/helpers/SwapHandler/Implementaions/LifiImplementation.sol

Status: Fixed

Recommendations

Remediation: Consider using `Ownable2Step` instead of `Ownable` from OpenZeppelin Contracts to enhance the security of your contract ownership management. This contract prevents the accidental transfer of ownership to an address that cannot handle it, such as due to a typo, by requiring the recipient of owner permissions to actively accept ownership via a contract call. This two-step ownership transfer process adds an additional layer of security to your contract's ownership management.

Remediation (revised commit: bee7c27): `Ownable2Step` was implemented as recommended.

F-2024-1430 - Missing events emitting for critical functions - Info

Description: Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes with timelocks that allow users to evaluate them and consider if they would like to engage/exit based on how they perceive the changes as affecting the trustworthiness of the protocol or profitability of the implemented financial services. The alternative of directly querying the on-chain contract state for such changes is not considered practical for most users/usages.

Missing events do not promote transparency and if such changes immediately affect users' perception of fairness or trustworthiness, they could exit the protocol causing a reduction in liquidity which could negatively impact protocol TVL and reputation.

Assets:

- contracts/accountingManager/AccountingManager.sol
- contracts/accountingManager/registry.sol
- contracts/governance/Keepers.sol
- contracts/helpers/BaseConnector.sol
- contracts/helpers/OmniChainHandler/OmnichainLogic.sol
- contracts/helpers/SwapHandler/GenericSwapAndBridgeHandler.sol
- contracts/helpers/SwapHandler/Implementaions/LifiImplementation.sol
- contracts/helpers/valueOracle/NoyaValueOracle.sol
- contracts/helpers/valueOracle/oracles/ChainlinkOracleConnector.sol

Status: Fixed

Recommendations

Remediation: Consider emitting the corresponding events in the critical functions.

Remediation (revised commit: 92df0dc): The event emitting has been added to the aforementioned assets

F-2024-1714 - Default compiler behavior leads to the gas increase in the for loop - Info

Description:	Newer versions of the Solidity compiler will check for integer overflows and underflows automatically. This provides safety but increases gas costs. When an unsigned integer is guaranteed to never overflow, the unchecked feature of Solidity can be used to save gas costs. A common case for this is for-loops using a strictly less than comparison in their conditional statement.
Assets:	• contracts/governance/Keepers.sol • contracts/helpers/SwapHandler/GenericSwapAndBridgeHandler.sol
Status:	Fixed

Recommendations

Remediation:	It is recommended to use an unchecked{++i;} operator to disable the default compiler checks and save gas. **Remediation (revised commit: bee7c27):** the `unchecked` operator was implemented as recommended.

F-2024-1770 - Improper threshold violates the requirements of multi-signature wallet - Info

Description:

According to the definition of the multi-signature wallet, any transaction must be signed by two or more signers. However, the current implementation of the `Keepers` contract allows to set a `threshold` to 1, which means that the number of signatures that will be used, will equal 1. The lack of validation can be seen in the code snippets below.

```
constructor(address[] memory _owners, uint8 _threshold) EIP712("Keepers", "1") Ow
nable(msg.sender) {
require(_owners.length <= 10 && _threshold <= _owners.length && _threshold != 0);
for (uint256 i = 0; i < _owners.length; i++) {
isOwner[_owners[i]] = true;
}
numOwners = _owners.length;
threshold = _threshold;
}
```

```
function updateOwners(address[] memory _owners, bool[] memory addOrRemove) public
onlyOwner {
uint256 numOwnersTemp = numOwners;
for (uint256 i = 0; i < _owners.length; i++) {
if (addOrRemove[i] && !isOwner[_owners[i]]) {
isOwner[_owners[i]] = true;
numOwnersTemp++;
} else if (!addOrRemove[i] && isOwner[_owners[i]]) {
isOwner[_owners[i]] = false;
numOwnersTemp--;
}
}
require(numOwnersTemp <= 10 && threshold <= numOwnersTemp && threshold != 0);
numOwners = numOwnersTemp;
}
```

and

```
function setThreshold(uint8 _threshold) public onlyOwner {
require(_threshold <= numOwners && _threshold != 0);
threshold = _threshold;
}
```

Assets:

- contracts/governance/Keepers.sol

Status: Fixed

Recommendations

Remediation:

It is recommended to ensure that the threshold of the Keepers contract will be bigger than `1` to not violate the requirements of the multi-signature wallet.

Remediation (revised commit: bee7c27): a check was introduced into the `constructor`, `updateOwners` and `setThreshold` methods to ensure the threshold is bigger than `1` in order to comply with multi-signature.

F-2024-1800 - Missing onERC721Received callback creates risks - Info

Description: To deal with ERC721 tokens securely, contracts or recipients should implement the IERC721Receiver interface by implementing the `onERC721Received` callback function. The callback is called every time the ERC721 token is transferred via the `safeTransferFrom()`. It is a way of signaling back to the `safeTransferFrom()`, that the recipient understands that it can deal with ERC721 and should implement the necessary functionality to handle such tokens. Even though the previously mentioned callback function cannot guarantee the safety of NFTs, it is recommended to be followed.

Assets:

- contracts/connectors/UNIv3Connector.sol

Status: Mitigated

Recommendations

Remediation: It is recommended to implement `onERC721Received` in all the contracts that have to deal with NFT tokens.

Remediation: this issue was determinted `Mitigated` with the following feedback provided by the Noya team:

> Although this contract is working with NFT positions, it's not going to use (both as the sender and receiver) the safeTransferFrom function. It's using NonfungiblePositionManager that is handling the positions. And it's using the _mint function which doesn't require onERC721Received.
> Any contract that is inheriting from this contract (just like the pancakeSwap example) can add this function for specific use cases. but it's not needed in UNIv3Connector.

F-2024-1801 - Performance fees only works on increasing vault valuation - Info

Description:

Due to the following check in `recordProfitForFee`, the performance fee will only work when the vault underlying tokens valuation increases:

```
if (storedProfitForFee < totalProfitCalculated) {
return;
}
```

The reason is that, when calling `collectPerformanceFees`, the previous value of `storedProfitForFee` is assigned to `totalProfitCalculated` and, therefore, only increasing `storedProfitForFee` will pass the check.

Assets:

- contracts/accountingManager/AccountingManager.sol

Status: Mitigated

Recommendations

Remediation:

Consider if this is the intended behaviour of the system.

Remediation: this issue was marked as `Mitigated` after Noya's team feedback below:

> This is intended because the receiver is eligible for performance fee if the profit has increased.

F-2024-1827 - Loop for storage array length wastes extra Gas - Info

Description: In `removeTrustedPosition`, a loop iterates through the storage array length `vault.holdingPositions.length`, wasting an unnecessary amount of Gas.

Since every iteration will read the storage variable to calculate the length, the amount of Gas can be reduced by caching the length into a new memory variable to be iterated.

Assets:

- contracts/accountingManager/registry.sol

Status: Fixed

Recommendations

Remediation: It is recommended to cache the array length `vault.holdingPositions.length` into a new memory variable to be iterated.

Remediation (revised commit: bee7c27): the `vault.holdingPositions.length` was cached into a new memory variable `length` and used within the loop as recommended.

F-2024-1840 - The direct usage of the Aerdrome Pool's functions leads to the extra gas and creates security risks - Info

Description:

Direct usage of the `Pool.mint()` and `Pool.burn()` functions causes a risk of frontrunning. Furthermore, the `skim()` function, which is used to get the stuck ERC20 tokens from the pool contract, costs additional gas. It can be seen in the code snippet below.

```
function supply(address pool, uint256 amount0, uint256 amount1, uint256 minBalanc
e) public onlyManager {
bytes32 positionId = registry.calculatePositionId(address(this), AERODROME_POSITI
ON_TYPE, abi.encode(pool));
IERC20(IPool(pool).token0()).safeTransfer(pool, amount0);
IERC20(IPool(pool).token1()).safeTransfer(pool, amount1);
uint256 liquidity = IPool(pool).mint(address(this));
registry.updateHoldingPosition(vaultId, positionId, "", "", false);
IPool(pool).skim(address(this));
_updateTokenInRegistry(IPool(pool).token0());
_updateTokenInRegistry(IPool(pool).token1());
if (liquidity < minBalance) revert IConnector_InvalidAmount();
}
```

The following approach cannot be considered a best practice due to the previously mentioned reasons.

Assets:

- contracts/connectors/AerodromeConnector.sol

Status: Fixed

Recommendations

Remediation:

It is recommended to directly interact with the Router contract.

Remediation (revised commit: 4847503): The `AerodromeConnector` now uses `IRouter` to add and remove liquidity in the `AerodromeConnector::supply()` and `AerodromeConnector::withdraw()` functions instead of the direct usage of the `IPool.mint()` and `IPool.burn()`.

F-2024-1844 - Checks-Effects-Interactions Pattern Violation - Info

Description:

State variables are updated after the external calls to the token contract.

As explained in Solidity Security Considerations, it is best practice to follow the checks-effects-interactions pattern when interacting with external contracts to avoid reentrancy-related issues.

The following assets are affected by this pattern violation:

`contracts/accountingManager/AccountingManager.sol: deposit, executeDeposit, retrieveTokensFroWithdraw, recordProfitForFee, checkIfTVLHasDroped.`
`contracts/accountingManager/registry.sol: addTrustedPosition.`
`contracts/helpers/OmniChainHandler/OmnichainLogic.sol: startBridgeTransaction.`
`contracts/helpers/SwapHandler/GenericSwapAndBridgeHandler.sol: executeSwap.`
`contracts/helpers/BaseConnector.sol: sendTokensToTrustedAddress, addLiquidity, swapHoldings.`
`contracts/connectors/BalancerFlashLoan.sol: _paybackFlashLoan, receiveFlashLoan.`
`contracts/connectors/CurveConnector.sol: openCurvePosition, decreaseCurvePosition, harvestRewards, harvestPrismaRewards, harvestConvexRewards.`
`contracts/connectors/BalancerConnector.sol: harvestAuraRewards, openPosition, decreasePosition.`
`contracts/connectors/FraxConnector.sol: borrowAndSupply, withdraw, repay.`
`contracts/connectors/GearBoxV3.sol: closeAccount, executeCommands.`
`contracts/connectors/LidoConnector.sol: deposit, requestWithdrawals, claimWithdrawal.`
`contracts/connectors/MaverickConnector.sol: stake, unstake, addLiquidityInMaverickPool, removeLiquidityFromMaverickPool.`
`contracts/connectors/PendleConnector.sol: supply, mintPTAndYT, depositIntoMarket, swapExactPTForSY, burnLP, decreasePosition, claimRewards.`
`contracts/connectors/AaveConnector.sol: supply, borrow, repay, withdrawCollateral.`
`contracts/connectors/AerodromeConnector.sol: supply, withdraw.`
`contracts/connectors/CompoundConnector.sol: supply, withdrawOrBorrow, claimRewards.`
`contracts/connectors/PancakeswapConnector.sol: updatePosition, withdraw.`
`contracts/connectors/PrismaConnector.sol: openTrove, adjustTrove, closeTrove.`
`contracts/connectors/SiloConnector.sol: deposit, withdraw, borrow, repay.`
`contracts/connectors/UNIv3Connector.sol: openPosition, decreasePosition, increasePosition, collectAllFees.`
`contracts/connectors/StargateConnector.sol: depositIntoStargatePool, withdrawFromStargatePool, claimStargateRewards.`

Status: Mitigated

Recommendations

Remediation:

Follow the checks-effects-interactions pattern when interacting with external contracts. If this approach is not possible, consider using a reentrancy guard.

Remediation (revised commit 119f7f3): the OpenZeppelin nonReentrant modifier was added to all reported functions with the exception of `BaseConnector::sendTokensToTrustedAddresses` and `BalancerFlashLoan::receiveFlashLoan` which would break the logic of the contract as explained by the Noya team. Due to the presence of access control mechanisms in both functions that limit the access to those functions, this issue was Mitigated, and the adequate usage of those functions relays on the Noya protocol.

F-2024-1845 - High oracle refresh check can lead to outdated pricing - Info

Description:	The age threshold used for the chainlink oracle is set to `5 days` by default. Given the case that many chainlink data feeds provide data in intervals close to `2h`, it is possible that assets will provide outdated data after 5 days. **Remediation (revised commit: bee7c27):** the default refresh rate `chainlinkPriceAgeThreshold` was set to `2 hours`. Additionally, the function `updateChainlinkPriceAgeThreshold` is implemented in order to update the refresh rate when necessary.
Assets:	• contracts/helpers/valueOracle/oracles/ChainlinkOracleConnector.sol
Status:	Fixed

Recommendations

Remediation:	Consider decreasing `chainlinkPriceAgeThreshold` to values closer than the data feed provider.

F-2024-1849 - Unused return value - Info

Description: The function `updateHoldingPosition` performs a call to `isPositionTrustedForConnector` but does not use it's return value.

Assets:

- contracts/accountingManager/registry.sol

Status: Fixed

Recommendations

Remediation: Use the return value of `isPositionTrustedForConnector`.

Remediation (revised commit: bee7c27): the return value of `isPositionTrustedForConnector` is now implemented as a safe check into `updateHoldingPosition`.

F-2024-1852 - Missing argument names in function - Info

Description: The function `sendTokensToTrustedAddress` uses several unnamed arguments.

```
function sendTokensToTrustedAddress(address token, uint256 amount, address, bytes
calldata)
```

Assets:

- contracts/accountingManager/AccountingManager.sol

Status: Fixed

Recommendations

Remediation: It is recommended to name all function arguments.

Remediation (revised commit: bee7c27): the reported function arguments were named as recommended.

F-2024-1853 - Sub-optimal management of fees values - Info

Description: The fees implemented in the vaults (i.e. `withdrawFee`, `performanceFees` and `managementFee`) are not implemented in the `constructor`. Therefore, functions using fees may be used without the required values.

In addition, the values that can be set via `setFees` are not limited, allowing for up to `100%` fees. This allows abusive behaviour and should not be allowed.

```
function setFees(uint256 _withdrawFee, uint256 _performanceFee, uint256 _manageme
ntFee) public onlyMaintainer {
withdrawFee = _withdrawFee;
performanceFee = _performanceFee;
managementFee = _managementFee;
emit FeeRatesChanged(_withdrawFee, _performanceFee, _managementFee);
}
```

Assets:

- contracts/accountingManager/AccountingManager.sol

Status: Fixed

Recommendations

Remediation: Call `setFees` in the `constructor` and implement a limit to the values they can reach.

Remediation (revised commit: bee7c27): the fee values are now set within the `constructor`. Additionally, maximum fee values were implemented as the constants `WITHDRAWAL_MAX_FEE`, `MANAGEMENT_MAX_FEE` and `PERFORMANCE_MAX_FEE`. A check was introduced in both `constructor` and `setFees` methods to make sure the fees cannot surpass their maximum value.

F-2024-1854 - Non-standard filename - Info

Description:	The file `registry.sol` is not written in the recommended manner: `Registry.sol`.
Assets:	• contracts/accountingManager/registry.sol
Status:	Fixed

Recommendations

Remediation:	Rename `registry.sol` to `Registry.sol`. **Remediation (revised commit: bee7c27):** the reported file was renamed `Registry.sol` as recommended.

F-2024-1855 - Missing and incorrect NatSpec - Info

Description: The project NatSpec, implemented for both contracts and functions, is not sufficient. Most of the functions do not have a NatSpec, or have an oldest NatSpec, or have an incorrect NatSpec.

Status: Accepted

Recommendations

Remediation: Implement an up-to-date NatSpec in all contracts and functions.

Remediation (revised commit: bee7c27): the NatSpec was improved but it is still limited. Contracts such as `NoyaTimeLock` and `Watchers` and `WETH_Oracle` do not have any NatSpec. The majority of the contracts do not have a contract description NatSpec. The connectors have a superficial NatSpec that describes the function name, but not how the function works.

F-2024-1856 - Incorrect return value - Info

Description:

The function `updateHoldingPosition` returns `100` when a holding position is removed. However, this value is not checked or has any use in the protocol.

```
function updateHoldingPosition() public vaultExists(vaultId) returns (uint256) {
...
if (removePosition) {
if (positionIndex < vault.holdingPositions.length - 1) {
vault.holdingPositions[positionIndex] = vault.holdingPositions[vault.holdingPosit
ions.length - 1];
vault.isPositionUsed[keccak256(
abi.encode(
vault.holdingPositions[positionIndex].calculatorConnector,
vault.holdingPositions[positionIndex].positionId,
vault.holdingPositions[positionIndex].data
)
)] = positionIndex;
}
vault.holdingPositions.pop();
vault.isPositionUsed[holdingPositionId] = 0;
emit HoldingPositionUpdated(vaultId, _positionId, _data, additionalData, removePo
sition, positionIndex);
return 100;
}
...
}
```

This scenario is present in both `registry` and `GearBoxV3` contracts.

```
function _getPositionTVL(HoldingPI memory p, address base) public view override r
eturns (uint256 tvl) {
...
if (d.totalDebtUSD > d.totalValueUSD) {
return 100;
}
return _getValue(address(840), base, (d.totalValueUSD - d.totalDebtUSD));
}
```

Assets:

- contracts/accountingManager/registry.sol

Status: Fixed

Recommendations

Remediation: Consider deleting the return value `100`.

Remediation (revised commit: bee7c27): the function `_getPositionTVL` now returns `0` since the reported branch should not be executed. The function `updateHoldingPosition` now returns `type(uint256).max`, indicating that the position doesn't exist anymore in the array.

F-2024-1858 - Multiplication after division may result in precision loss - Info

Description:

Due to Solidity characteristic rounding values, it is not recommended to perform multiplications after divisions since it can lead to loss of precision.

This is present in `_getHealthFactor` for Frax connector and in `getCollBlanace` for Compound connector,

```
function _getHealthFactor(IFraxPair _fraxlendPair, uint256 _exchangeRate) interna
l view virtual returns (uint256) {
...
uint256 currentPositionLTV =
(((_borrowerAmount * _exchangeRate) / EXCHANGE_PRECISION) * LTV_PRECISION) / _col
lateralAmount;
...
}
```

```
function getCollBlanace(IComet comet, bool riskAdjusted) public view returns (uin
t256 CollValue) {
...
uint256 collateralValueInVirtualBase =
collateralBalance * collateralPriceInVirtualBase / info.scale * comet.baseScale()
/ basePrice;
...
}
```

Assets:

- contracts/connectors/CompoundConnector.sol
- contracts/connectors/FraxConnector.sol

Status: Fixed

Recommendations

Remediation:

Perform multiplications before divisions to avoid precision loss.

Remediation (revised commit: bee7c27): the reported functions were updated so that multiplications are performed before of divisions, avoiding precision loss.

F-2024-1861 - Missing expiration timestamp can lead to unexpected results - Info

Description:

Advanced protocols like Automated Market Makers (AMMs) can allow users to specify a deadline parameter that enforces a time limit by which the transaction must be executed. Without a deadline parameter, the transaction may sit in the mempool and be executed at a much later time potentially resulting in a worse price for the user.

Protocols shouldn't set the deadline to `block.timestamp` as a validator can hold the transaction and the block it is eventually put into will be `block.timestamp`, so this offers no protection.

The transaction deadline was set to `block.timestamp` in the functions that can be seen in the code snippets below.

`UNIv3Connector::openPosition()`

```
function openPosition(MintParams memory p) external onlyManager returns (uint256
tokenId) {
bytes32 positionId =
registry.calculatePositionId(address(this), UNI_LP_POSITION_TYPE, abi.encode(p.to
ken0, p.token1));
p.recipient = address(this);
p.deadline = block.timestamp;
// Approve NonfungiblePositionManager to spend `token0` and `token1`.
_approveOperations(p.token0, address(positionManager), p.amount0Desired);
_approveOperations(p.token1, address(positionManager), p.amount1Desired);

// Supply liquidity to pool.
(tokenId,,,) = positionManager.mint(p);
bytes memory positionData = abi.encode(tokenId);
registry.updateHoldingPosition(
vaultId, positionId, positionData, abi.encode(p.tickLower, p.tickUpper, p.fee), f
alse
);
_updateTokenInRegistry(p.token0);
_updateTokenInRegistry(p.token1);
}
```

`MaverickConnector::addLiquidityInMaverickPool()`

```
function addLiquidityInMaverickPool(MavericAddLiquidityParams calldata p) externa
l onlyManager {
uint256 sendEthAmount = p.ethPoolIncluded ? p.tokenARequiredAllowance : 0;
_approveOperations(p.pool.tokenA(), maverickRouter, p.tokenARequiredAllowance); /
/ TODO: check token A is eth
_approveOperations(p.pool.tokenB(), maverickRouter, p.tokenBRequiredAllowance);
// add liquidity
uint256 tokenId;
{
(tokenId,,,) = IMaverickRouter(maverickRouter).addLiquidityToPool{ value: sendEth
Amount }(
p.pool, 0, p.params, p.minTokenAAmount, p.minTokenBAmount, block.timestamp
);
}
registry.updateHoldingPosition(
vaultId, registry.calculatePositionId(address(this), MAVERICK_LP, abi.encode(p.po
ol)), "", "", false
);
_updateTokenInRegistry(p.pool.tokenA());
_updateTokenInRegistry(p.pool.tokenB());
}
```

`MaverickConnector::removeLiquidityFromMaverickPool()`

```
function removeLiquidityFromMaverickPool(
IMaverickPool pool,
uint256 tokenId,
RemoveLiquidityParams[] calldata params,
uint256 minTokenAAmount,
uint256 minTokenBAmount
) external {
IMaverickPosition position = IMaverickRouter(maverickRouter).position();
position.approve(maverickRouter, tokenId);
IMaverickRouter(maverickRouter).removeLiquidity(
pool, address(this), tokenId, params, minTokenAAmount, minTokenBAmount, block.tim
estamp
);
registry.updateHoldingPosition(
vaultId, registry.calculatePositionId(address(this), MAVERICK_LP, abi.encode(pool
)), "", "", true
);
_updateTokenInRegistry(pool.tokenA());
_updateTokenInRegistry(pool.tokenB());
}
```

Assets:

- contracts/connectors/MaverickConnector.sol
- contracts/connectors/UNIv3Connector.sol

Status:	Fixed

Recommendations

Remediation: Introduce a proper deadline parameter for the previously mentioned functions.

Remediation (revised commit: 774997c): The deadline is not hardcoded anymore and now is passed as a parameter to the functions `UNIv3Connector::openPosition()`, `MaverickConnector::addLiquidityInMaverickPool()` and `MaverickConnector::removeLiquidityFromMaverickPool()`.

F-2024-1863 - LifiImplementation contract is incompatible with non-standard ERC20 tokens - Info

Description: Some tokens (for example `USDT`) do not follow the ERC20 standard correctly and do not revert a `bool` on `approve` call. Such tokens are incompatible with the `LifiImplementation` contract as the `approve` calls in function `LifiImplementation::_setAllowance()` might revert or return bool values, which are not validated.

```
function _setAllowance(IERC20 token, address spender, uint256 amount) internal {
if (_isNative(token)) {
return;
}
if (spender == address(0)) {
revert SpenderIsInvalid();
}

uint256 allowance = token.allowance(address(this), spender);

if (allowance < amount) {
if (allowance != 0) {
token.approve(spender, 0);
}
token.approve(spender, type(uint256).max);
}
}
```

Status: Fixed

Recommendations

Remediation: Use SafeERC20's `forceApprove()` instead.

Remediation (revised commit: bee7c27): the SafeERC20's `forceApprove` was implemented as recommended.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details

Repository	https://github.com/Noya-ai/noya-vault-contracts
Audited commit	95fcb52
Remediation commit	6f42e71
Whitepaper	https://docs.noya.ai/
Requirements	https://github.com/Noya-ai/noya-vault-contracts/README.md
Technical Requirements	https://github.com/Noya-ai/noya-vault-contracts/README.md

Contracts in Scope

./contracts/accountingManager/AccountingManager.sol

./contracts/accountingManager/NoyaFeeReceiver.sol

./contracts/accountingManager/registry.sol

./contracts/governance/Keepers.sol

./contracts/governance/NoyaGovernanceBase.sol

./contracts/governance/TimeLock.sol

./contracts/governance/Watchers.sol

./contracts/helpers/LZHelpers/LZHelperReceiver.sol

./contracts/helpers/LZHelpers/LZHelperSender.sol

./contracts/helpers/OmniChainHandler/OmnichainLogic.sol

./contracts/helpers/OmniChainHandler/OmnichainManagerBaseChain.sol

./contracts/helpers/OmniChainHandler/OmnichainManagerNormalChain.sol

./contracts/helpers/SwapHandler/Implementaions/LifiImplementation.sol

./contracts/helpers/SwapHandler/GenericSwapAndBridgeHandler.sol

./contracts/helpers/valueOracle/oracles/ChainlinkOracleConnector.sol

./contracts/helpers/valueOracle/oracles/UniswapValueOracle.sol

./contracts/helpers/valueOracle/oracles/WETH_Oracle.sol

./contracts/helpers/valueOracle/NoyaValueOracle.sol

./contracts/helpers/BaseConnector.sol

./contracts/helpers/TVLHelper.sol

./contracts/connectors/BalancerFlashLoan.sol

./contracts/connectors/CurveConnector.sol

./contracts/connectors/BalancerConnector.sol

./contracts/connectors/FraxConnector.sol

./contracts/connectors/GearBoxV3.sol

./contracts/connectors/LidoConnector.sol

Contracts in Scope

./contracts/connectors/MaverickConnector.sol

./contracts/connectors/MorphoBlueConnector.sol

./contracts/connectors/PendleConnector.sol

./contracts/connectors/AaveConnector.sol

./contracts/connectors/AerodromeConnector.sol

./contracts/connectors/CompoundConnector.sol

./contracts/connectors/PancakeswapConnector.sol

./contracts/connectors/PrismaConnector.sol

./contracts/connectors/SiloConnector.sol

./contracts/connectors/UNIv3Connector.sol

./contracts/connectors/StargateConnector.sol