Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

2 Audit Methodology

3 Project Overview

3.1 Project Introduction

3.2 Vulnerability Information

4 Code Overview

4.1 Contracts Description

4.2 Visibility Description

4.3 Vulnerability Summary

5 Audit Result

6 Statement

1 Executive Summary

On 2023.03.29, the SlowMist security team received the Narwhal Finance team's security audit application for Narwhal Finance, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

Narwhal Finance is a decentralized perpetual trading platform. It allows users to trade a wide range of leverages and pairs (Crypto and FX) without KYC, geographic restrictions, or high centralized exchange fees. The protocol ecosystem revolves around USDT, a stablecoin that can be used as collateral when opening trades; NLP, the liquidity token used as a counterparty to traders; and NAR, the protocol’s governance, and utility token.

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Missing event record	Others	Suggestion	Acknowledged
N2	Lack of judgment for newTP	Design Logic Audit	Medium	Fixed
N3	function to repeat the effect	Gas Optimization Audit	Suggestion	Fixed
N4	Redundant SafeMath is used	Gas Optimization Audit	Suggestion	Acknowledged
N5	Potential incorrect decimal record issue	Arithmetic Accuracy Deviation Vulnerability	High	Fixed
N6	USDT Token Transfer Compatibility Issue	Others	Critical	Fixed
N7	Redundant pause function	Gas Optimization Audit	Suggestion	Fixed
N8	Incorrect use of nonReentrant modifier	Gas Optimization Audit	Suggestion	Fixed
N9	Potential non-full vesting of tokens	Design Logic Audit	Medium	Fixed
N10	Risk of low-cost rewards through front-running	Design Logic Audit	Low	Fixed
N11	Missing return value check	Others	Suggestion	Fixed
N12	Potential economic risks of the Vesting lock model	Design Logic Audit	Low	Fixed
N13	Redundant unstake logic issue	Others	Suggestion	Fixed
N14	storageT cannot be changed	Others	Suggestion	Acknowledged
N15	Authority transfer enhancement	Others	Low	Fixed
N16	Precision compatibility issues	Arithmetic Accuracy	High	Fixed

NO	Title	Category	Level	Status
		Deviation Vulnerability		
N17	Business logic is unclear	Others	Suggestion	Fixed
N18	Lack of same bots judgment	Design Logic Audit	Low	Acknowledged
N19	Compatibility risk of updateReward and allowed features	Design Logic Audit	Critical	Fixed
N20	referrerDetails record issue	Design Logic Audit	Low	Fixed
N21	Lack of initialization parameter validation	Design Logic Audit	Medium	Fixed
N22	USDT decimal compatibility issue	Arithmetic Accuracy Deviation Vulnerability	Critical	Fixed
N23	Open a position without updating the price to increase the winning rate	Design Logic Audit	High	Fixed
N24	Incorrect rewards account acquisition	Design Logic Audit	Critical	Fixed
N25	Potential risk of missing modifiers	Others	Medium	Fixed
N26	Risk of not being able to lock	Design Logic Audit	Low	Acknowledged
N27	Minimum USDT position check issue	Design Logic Audit	Suggestion	Acknowledged
N28	Missing check for maximum PnL when updating Take Profit	Design Logic Audit	High	Fixed
N29	Redundant temp variable in closeTradeMarket	Others	Suggestion	Fixed
N30	`getPrice` missing check for `UPDATE_SL` type	Others	Suggestion	Acknowledged

NO	Title	Category	Level	Status
N31	Risk of excessive permissions	Authority Control Vulnerability Audit	Medium	Acknowledged
N32	Time check inconsistency	Design Logic Audit	Low	Fixed

4 Code Overview

4.1 Contracts Description

Codebase:

Audit Version:

https://github.com/Narwhal-Finance/narwhal-contracts-QA

commit: 68cb322c2f6f0d16752063feb46f07a29ee6fae4

Fixed Version:

https://github.com/Narwhal-Finance/narwhal-contracts-QA

commit: 02bbe6c5dbf4f009c92a94d99e010aeff4a17909

Audit Scope:

- contracts/Tokens/BaseToken.sol
- contracts/Tokens/Narwhal token.sol
- contracts/Tokens/Vester.sol
- contracts/Tokens/VesterNLP.sol
- contracts/Tokens/VestingSchedule.sol
- contracts/Tokens/esNAR.sol
- contracts/LimitOrdersStorage.sol
- contracts/NarwhalPool.sol
- contracts/NarwhalPriceAggregator.sol
- contracts/NarwhalReferrals.sol

- contracts/NarwhalTrading.sol
- contracts/NarwhalTradingCallbacks.sol
- contracts/PairInfos.sol
- contracts/PairsStorage.sol
- contracts/TradingStorage.sol
- contracts/TradingVaultV2.sol

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

BaseToken			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setGov	External	Can Modify State	onlyGov
setInfo	External	Can Modify State	onlyGov
addAdmin	External	Can Modify State	onlyGov
removeAdmin	External	Can Modify State	onlyGov
withdrawToken	External	Can Modify State	onlyGov
setInPrivateTransferMode	External	Can Modify State	onlyGov
setHandler	External	Can Modify State	onlyGov
addNonStakingAccount	External	Can Modify State	onlyAdmin
removeNonStakingAccount	External	Can Modify State	onlyAdmin
totalStaked	External	-	-
balanceOf	External	-	-

BaseToken

stakedBalance	External	-	-
transfer	External	Can Modify State	-
allowance	External	-	-
approve	External	Can Modify State	-
transferFrom	External	Can Modify State	-
_mint	Internal	Can Modify State	-
_burn	Internal	Can Modify State	-
_transfer	Private	Can Modify State	-
_approve	Private	Can Modify State	-

MintableBaseToken

Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	BaseToken
setMinter	External	Can Modify State	onlyGov
mint	External	Can Modify State	onlyMinter
burn	External	Can Modify State	onlyMinter

esNAR

Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	MintableBaseToken
id	External	-	-

Governable

Function Name	Visibility	Mutability	Modifiers

Governable			
<Constructor>	Public	Can Modify State	-
setGov	External	Can Modify State	onlyGov

Vester			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setHandler	External	Can Modify State	onlyGov
setNarwhalPool	Public	Can Modify State	onlyGov
setHasMaxVestableAmount	External	Can Modify State	onlyGov
setTransferredAverageStakedAmounts	External	Can Modify State	nonReentrant
settransferredCumulativeRewards	External	Can Modify State	nonReentrant
claimable	Public	-	-
getMaxVestableAmount	Public	-	-
getPairAmount	Public	-	-
getCombinedAverageStakedAmount	Public	-	-
getTotalVested	Public	-	-
balanceOf	Public	-	-
transfer	Public	-	-
allowance	Public	-	-
approve	Public	Can Modify State	-
transferFrom	Public	Can Modify State	-
getVestedAmount	Public	-	-
deposit	External	Can Modify State	nonReentrant

Vester			
depositForAccount	External	Can Modify State	nonReentrant
_deposit	Private	Can Modify State	-
withdrawToken	External	Can Modify State	onlyGov
withdraw	External	Can Modify State	nonReentrant
claim	External	Can Modify State	nonReentrant
claimForAccount	External	Can Modify State	nonReentrant
_claim	Private	Can Modify State	-
_mint	Private	Can Modify State	-
_burn	Private	Can Modify State	-
_mintPair	Private	Can Modify State	-
_burnPair	Private	Can Modify State	-
_updateVesting	Private	Can Modify State	-
_getNextClaimableAmount	Private	-	-
_validateHandler	Private	-	-

VesterNLP			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setHandler	External	Can Modify State	onlyGov
setTradingVault	Public	Can Modify State	onlyGov
setHasMaxVestableAmount	External	Can Modify State	onlyGov
setTransferredAverageStakedAmounts	External	Can Modify State	nonReentrant
settransferredCumulativeRewards	External	Can Modify State	nonReentrant

VesterNLP			
claimable	Public	-	-
getMaxVestableAmount	Public	-	-
getPairAmount	Public	-	-
getCombinedAverageStakedAmount	Public	-	-
getTotalVested	Public	-	-
balanceOf	Public	-	-
transfer	Public	-	-
allowance	Public	-	-
approve	Public	Can Modify State	-
transferFrom	Public	Can Modify State	-
getVestedAmount	Public	-	-
deposit	External	Can Modify State	nonReentrant
depositForAccount	External	Can Modify State	nonReentrant
_deposit	Private	Can Modify State	-
withdrawToken	External	Can Modify State	onlyGov
withdraw	External	Can Modify State	nonReentrant
claim	External	Can Modify State	nonReentrant
claimForAccount	External	Can Modify State	nonReentrant
_claim	Private	Can Modify State	-
_mint	Private	Can Modify State	-
_burn	Private	Can Modify State	-
_mintPair	Private	Can Modify State	-
_burnPair	Private	Can Modify State	-

VesterNLP			
_updateVesting	Private	Can Modify State	-
_getNextClaimableAmount	Private	-	-
_validateHandler	Private	-	-

VestingSchedule			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
start	Public	Can Modify State	onlyOwner
changeClaimerStatus	Public	Can Modify State	onlyOwner
addGroupCliff	Public	Can Modify State	onlyOwner
transferAnyERC20	Public	Can Modify State	onlyOwner
balanceOf	Public	-	-
addAmountToInvestors	Public	Can Modify State	onlyOwner
claimable	Public	-	-
claim	External	Can Modify State	nonReentrant
_claim	Private	Can Modify State	-
_mint	Private	Can Modify State	-
_burn	Private	Can Modify State	-
_updateVesting	Private	Can Modify State	-
_getNextClaimableAmount	Private	-	-
transfer	Public	-	-
allowance	Public	-	-
approve	Public	Can Modify State	-

VestingSchedule			
transferFrom	Public	Can Modify State	-

NarwhalPool			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setTokenWhitelisted	Public	Can Modify State	onlyGov
setRewardsDuration	External	Can Modify State	onlyGov
notifyRewardAmount	External	Can Modify State	onlyGov updateReward
setGovFund	External	Can Modify State	onlyGov
setVester	External	Can Modify State	onlyGov
setEsToken	External	Can Modify State	onlyGov
setNARToken	External	Can Modify State	onlyGov
setUSDT	External	Can Modify State	onlyGov
setTradingVault	External	Can Modify State	onlyGov
addAllowedContract	External	Can Modify State	onlyGov
removeAllowedContract	External	Can Modify State	onlyGov
increaseAccTokens	External	Can Modify State	-
totalSupply	External	-	-
balanceOf	External	-	-
lastTimeRewardApplicable	Public	-	-
rewardPerToken	Public	-	-
earned	Public	-	-
getRewardForDuration	External	-	-

NarwhalPool			
getUserTokenBalance	Public	-	-
getReservedAmount	Public	-	-
pendingRewardUSDTNARStake	Public	-	-
getUserNARInfo	Public	-	-
_lockNAR	Public	Can Modify State	-
_unLockNAR	Public	Can Modify State	-
harvest	Public	Can Modify State	updateReward
_harvest	Internal	Can Modify State	-
stake	Public	Can Modify State	nonReentrant updateReward
unstake	Public	Can Modify State	nonReentrant updateReward
userAvailToWithdraw	Public	-	-
userTotalBalance	Public	-	-

NarwhalTradingVault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC20
setRewardsDuration	External	Can Modify State	onlyGov
notifyRewardAmount	External	Can Modify State	onlyGov updateReward
setAllowed	Public	Can Modify State	onlyOwner
balance	Public	-	-
getUserTokenBalance	Public	-	-
getPricePerFullShare	Public	-	-
lastTimeRewardApplicable	Public	-	-

NarwhalTradingVault

rewardPerToken	Public	-	-
earned	Public	-	-
getRewardForDuration	External	-	-
setRewardToken	External	Can Modify State	onlyGov
setVester	External	Can Modify State	onlyGov
setWithdrawTimelock	External	Can Modify State	onlyGov
_lockNAR	Public	Can Modify State	-
_unLockNAR	Public	Can Modify State	-
harvest	Public	Can Modify State	updateReward
depositAll	External	Can Modify State	-
deposit	Public	Can Modify State	updateReward nonReentrant
withdrawAll	External	Can Modify State	-
withdraw	Public	Can Modify State	updateReward nonReentrant
userTotalBalance	Public	-	-
distributeRewardUSDT	Public	Can Modify State	onlyCallbacks
sendUSDTToTrader	External	Can Modify State	onlyCallbacks
claimUSDT	External	Can Modify State	-
receiveUSDTFromTrader	External	Can Modify State	onlyCallbacks

TradingStorage

Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setGov	External	Can Modify State	onlyGov

TradingStorage			
setDev	External	Can Modify State	onlyGov
addTradingContract	External	Can Modify State	onlyGov
removeTradingContract	External	Can Modify State	onlyGov
addSupportedToken	External	Can Modify State	onlyGov
setPriceAggregator	External	Can Modify State	onlyGov
setPool	External	Can Modify State	onlyGov
setVault	External	Can Modify State	onlyGov
setTrading	External	Can Modify State	onlyGov
setCallbacks	External	Can Modify State	onlyGov
setTokens	External	Can Modify State	onlyGov
setMaxTradesPerBlock	External	Can Modify State	onlyGov
setMaxTradesPerPair	External	Can Modify State	onlyGov
setMaxPendingMarketOrders	External	Can Modify State	onlyGov
setMaxGainP	External	Can Modify State	onlyGov
setDefaultLeverageUnlocked	External	Can Modify State	onlyGov
setMaxSlP	External	Can Modify State	onlyGov
setSpreadReductionsP	External	Can Modify State	onlyGov
setMaxOpenInterestUSDT	External	Can Modify State	onlyGov
storePendingNftOrder	External	Can Modify State	onlyTrading
unregisterPendingNftOrder	External	Can Modify State	onlyTrading
storeTrade	External	Can Modify State	onlyTrading
unregisterTrade	External	Can Modify State	onlyTrading
storePendingMarketOrder	External	Can Modify State	onlyTrading

TradingStorage			
unregisterPendingMarketOrder	External	Can Modify State	onlyTrading
updateOpenInterestUSDT	Private	Can Modify State	-
storeOpenLimitOrder	External	Can Modify State	onlyTrading
updateOpenLimitOrder	External	Can Modify State	onlyTrading
unregisterOpenLimitOrder	External	Can Modify State	onlyTrading
updateSl	External	Can Modify State	onlyTrading
updateTp	External	Can Modify State	onlyTrading
updateTrade	External	Can Modify State	onlyTrading
storeReferral	External	Can Modify State	onlyTrading
increaseReferralRewards	External	Can Modify State	onlyTrading
distributeLpRewards	External	Can Modify State	onlyTrading
setLeverageUnlocked	External	Can Modify State	onlyTrading
transferUSDT	External	Can Modify State	onlyTrading
firstEmptyTradeIndex	Public	-	-
firstEmptyOpenLimitIndex	Public	-	-
hasOpenLimitOrder	Public	-	-
getReferral	External	-	-
getLeverageUnlocked	External	-	-
pairTradersArray	External	-	-
getPendingOrderIds	External	-	-
pendingOrderIdsCount	External	-	-
getOpenLimitOrder	External	-	-

TradingStorage			
getOpenLimitOrders	External	-	-
getSupportedTokens	External	-	-
getSpreadReductionsArray	External	-	-

PairsStorage			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
changeStorageInterface	Public	Can Modify State	-
addPair	Public	Can Modify State	onlyGov feedOk groupListed feeListed
addPairs	External	Can Modify State	-
updatePair	External	Can Modify State	onlyGov feedOk feeListed
addGroup	External	Can Modify State	onlyGov groupOk
updateGroup	External	Can Modify State	onlyGov groupListed groupOk
addFee	External	Can Modify State	onlyGov feeOk
updateFee	External	Can Modify State	onlyGov feeListed feeOk
updateGroupCollateral	External	Can Modify State	-
pairJob	External	Can Modify State	-
pairFeed	External	-	-
pairSpreadP	External	-	-
pairMinLeverage	External	-	-
pairMaxLeverage	External	-	-

PairsStorage

groupMaxCollateral	External	-	-
groupCollateral	External	-	-
guaranteedSlEnabled	External	-	-
pairOpenFeeP	External	-	-
pairCloseFeeP	External	-	-
pairOracleFeeP	External	-	-
pairNftLimitOrderFeeP	External	-	-
pairReferralFeeP	External	-	-
pairMinLevPosUSDT	External	-	-
pairsBackend	External	-	-

PairInfos

Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setManager	External	Can Modify State	onlyGov
setMaxNegativePnlOnOpenP	External	Can Modify State	onlyManager
setPairParams	Public	Can Modify State	onlyManager
setPairParamsArray	External	Can Modify State	onlyManager
setOnePercentDepth	Public	Can Modify State	onlyManager
setOnePercentDepthArray	External	Can Modify State	onlyManager
setRolloverFeePerBlockP	Public	Can Modify State	onlyManager
setRolloverFeePerBlockPArray	External	Can Modify State	onlyManager
setFundingFeePerBlockP	Public	Can Modify State	onlyManager

PairInfos			
setFundingFeePerBlockPArray	External	Can Modify State	onlyManager
storeTradeInitialAccFees	External	Can Modify State	onlyCallbacks
storeAccRolloverFees	Private	Can Modify State	-
getPendingAccRolloverFees	Public	-	-
storeAccFundingFees	Private	Can Modify State	-
getPendingAccFundingFees	Public	-	-
getTradePriceImpact	External	-	-
getTradePriceImpactPure	Public	-	-
getTradeRolloverFee	Public	-	-
getTradeRolloverFeePure	Public	-	-
getTradeFundingFee	Public	-	-
getTradeFundingFeePure	Public	-	-
getTradeLiquidationPrice	External	-	-
getTradeLiquidationPricePure	Public	-	-
getTradeValue	External	Can Modify State	onlyCallbacks
getTradeValuePure	Public	-	-
getPairInfos	External	-	-
getOnePercentDepthAbove	External	-	-
getOnePercentDepthBelow	External	-	-
getRolloverFeePerBlockP	External	-	-
getFundingFeePerBlockP	External	-	-
getAccRolloverFees	External	-	-

PairInfos			
getAccRolloverFeesUpdateBlock	External	-	-
getAccFundingFeesLong	External	-	-
getAccFundingFeesShort	External	-	-
getAccFundingFeesUpdateBlock	External	-	-
getTradeInitialAccRolloverFeesPerCollateral	External	-	-
getTradeInitialAccFundingFeesPerOi	External	-	-
getTradeOpenedAfterUpdate	External	-	-

NarwhalTradingCallbacks			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
giveAllowance	Public	Can Modify State	onlyGov
openTradeMarketCallback	External	Can Modify State	onlyPriceAggregator
closeTradeMarketCallback	External	Can Modify State	onlyPriceAggregator
executeOpenOrderCallback	External	Can Modify State	onlyPriceAggregator
executeCloseOrderCallback	External	Can Modify State	onlyPriceAggregator
updateSlCallback	External	Can Modify State	onlyPriceAggregator
registerTrade	Private	Can Modify State	-
unregisterTrade	Internal	Can Modify State	-
withinExposureLimits	Internal	-	-
currentPercentProfit	Internal	-	-
correctTp	Internal	-	-
correctSl	Internal	-	-

NarwhalTradingCallbacks			
marketExecutionPrice	Internal	-	-

NarwhalTrading			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setMaxPosUSDT	External	Can Modify State	onlyGov
setLimitOrdersTimelock	External	Can Modify State	onlyGov
setAllowedToInteract	Public	Can Modify State	-
setVaultFactory	Public	Can Modify State	onlyGov
setMarketOrdersTimeout	External	Can Modify State	onlyGov
pause	External	Can Modify State	onlyGov
done	External	Can Modify State	onlyGov
executeNftOrder	External	Can Modify State	notContract notDone
openTrade	External	Can Modify State	-
closeTradeMarket	External	Can Modify State	notContract notDone
updateOpenLimitOrder	External	Can Modify State	notContract notDone
cancelOpenLimitOrder	External	Can Modify State	notContract notDone
updateTp	External	Can Modify State	notContract notDone
updateSl	External	Can Modify State	notContract notDone
getTradeLiquidationPrice	Private	-	-
openTradeMarketTimeout	External	Can Modify State	notContract notDone
closeTradeMarketTimeout	External	Can Modify State	notContract notDone

NarwhalReferrals			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getReferralDetails	Public	-	-
getReferralDiscountAndRebate	Public	-	-
isTier3KOL	Public	-	-
getReferral	Public	-	-
setBaseRebatesAndDiscounts	Public	Can Modify State	onlyOwner
setTier3Tier2RebateBonus	Public	Can Modify State	onlyOwner
setTradingVault	External	Can Modify State	onlyOwner
setWhitelistedAddress	Public	Can Modify State	onlyOwner
referKOLUnder	Public	Can Modify State	onlyOwner
incrementTier2Tier3	Public	Can Modify State	onlyCallbacks
incrementRewards	Public	Can Modify State	onlyCallbacks
claimRewards	Public	Can Modify State	-
changeReferralLink	Public	Can Modify State	nonReentrant
signUp	Public	Can Modify State	nonReentrant

NarwhalPriceAggregator			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
updatePairsStorage	External	Can Modify State	onlyGov
setUSDTFeed	Public	Can Modify State	onlyGov
setOracle	Public	Can Modify State	onlyGov

NarwhalPriceAggregator			
setAge	Public	Can Modify State	onlyGov
setNarwhalTrading	Public	Can Modify State	onlyGov
tokenPriceUSDT	Public	-	-
beforeGetPriceLimit	Public	Can Modify State	onlyTrading
getPrice	Public	Payable	onlyTrading
fulfill	Public	Can Modify State	-
getPythFee	Public	-	-
storePendingSlOrder	Internal	Can Modify State	-
unregisterPendingSlOrder	External	Can Modify State	-
swap	Private	-	-
sort	Private	-	-
median	Private	-	-
openFeeP	External	-	-
<Receive Ether>	External	Payable	-

LimitOrdersStorage			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
updateTriggerTimeout	External	Can Modify State	onlyGov
updateSameBlockLimit	External	Can Modify State	onlyGov
updatePercentages	External	Can Modify State	onlyGov
storeFirstToTrigger	External	Can Modify State	onlyTrading
storeTriggerSameBlock	External	Can Modify State	onlyTrading

LimitOrdersStorage			
unregisterTrigger	External	Can Modify State	onlyCallbacks
setOpenLimitOrderType	External	Can Modify State	onlyTrading
triggered	External	-	-
timedOut	External	-	-
sameBlockTriggers	External	-	-

4.3 Vulnerability Summary

[N1] [Suggestion] Missing event record

Category: Others

Content

In the BaseToken contract. The Gov role can set the sensitive parameters of the contract through the setGov, setInfo, addAdmin, removeAdmin, setInPrivateTransferMode and setHandler functions, but no event recording is performed.

The same is true for the setMinter function in the MintableBaseToken contract.

The same is true for the setHandler, setNarwhalPool, and setHasMaxVestableAmount functions in the Vester contract.

The same is true for the setTokenWhitelisted and setRewardsDuration functions in the NarwhalPool contract.

The same is true for the setAllowedToInteract and setVaultFactory functions in the NarwhalTrading contract.

The same is true for the setUSDTFeed, setOracle, setAge and setNarwhalTrading functions in the NarwhalPriceAggregator contract.

The same is true for the setBaseRebatesAndDiscounts setTier3Tier2RebateBonus, setTradingVault, setWhitelistedAddress and referKOLUnder functions in the NarwhalReferrals contract.

Code location:

contracts/Tokens/BaseToken.sol

```
    function setGov(address _gov) external onlyGov {
        gov = _gov;
    }

    function setInfo(
        string memory _name,
        string memory _symbol
    ) external onlyGov {
        name = _name;
        symbol = _symbol;
    }

    function addAdmin(address _account) external onlyGov {
        admins[_account] = true;
    }

    function removeAdmin(address _account) external override onlyGov {
        admins[_account] = false;
    }

    function setInPrivateTransferMode(
        bool _inPrivateTransferMode
    ) external override onlyGov {
        inPrivateTransferMode = _inPrivateTransferMode;
    }

    function setHandler(address _handler, bool _isActive) external onlyGov {
        isHandler[_handler] = _isActive;
    }
```

contracts/Tokens/esNAR.sol

```
    function setMinter(
        address _minter,
        bool _isActive
    ) external override onlyGov {
        isMinter[_minter] = _isActive;
    }
```

contracts/Token/Vester.sol

```
    function setHandler(address _handler, bool _isActive) external onlyGov {
        isHandler[_handler] = _isActive;
    }
    function setNarwhalPool(address _narwhalPool) public onlyGov {
        NarwhalPool = _narwhalPool;
    }

    function setHasMaxVestableAmount(
        bool _hasMaxVestableAmount
    ) external onlyGov {
        hasMaxVestableAmount = _hasMaxVestableAmount;
    }
```

contracts/NarwhalPool.sol

```
    function setTokenWhitelisted(address _token, bool _status) public onlyGov {
        isTokenWhitelisted[_token] = _status;
    }

    function setRewardsDuration(uint256 _rewardsDuration) external onlyGov {
        require(
            block.timestamp > periodFinish,
            "Previous rewards period must be complete before changing the duration 
for the new period"
        );
        rewardsDuration = _rewardsDuration;
    }
```

contracts/NarwhalTrading.sol

```
    function setAllowedToInteract(address _contract, bool _status) public {
        require(msg.sender == VaultFactory, "Not vault factory");
        allowedToInteract[_contract] = _status;
    }

    function setVaultFactory(address _VaultFactory) public onlyGov {
        require(_VaultFactory != address(0), "No dead address");
        VaultFactory = _VaultFactory;
    }
```

contracts/NarwhalPriceAggregator.sol

```
    function setUSDTFeed(bytes32 _feed) public onlyGov {
        USDTFeed = _feed;
    }

    function setOracle(address _oracle) public onlyGov {
        PythOracle = _oracle;
    }

    function setAge(uint256 _age) public onlyGov {
        require(_age <= 60, "Too much");
        age = _age;
    }

    function setNarwhalTrading(address _NarwhalTrading) public onlyGov {
        NarwhalTrading = _NarwhalTrading;
    }
```

contracts/NarwhalReferrals.sol

```
    function setBaseRebatesAndDiscounts(
        uint256 _discount,
        uint256 _rebate
    ) public onlyOwner {
        baseReferralDiscount = _discount;
        baseReferralRebate = _rebate;
    }

    function setTier3Tier2RebateBonus(
        uint256 _tier3tier2RebateBonus
    ) public onlyOwner {
        tier3tier2RebateBonus = _tier3tier2RebateBonus;
    }

    function setTradingVault(address _tradingVault) external onlyOwner {
        require(address(_tradingVault) != address(0), "ADDRESS_0");
        TradingVault = _tradingVault;
    }

    function setWhitelistedAddress(
        address _toWhitelist,
        bool _status,
        uint256 _rebate,
        uint256 _discount,
        uint256 _tier
    ) public onlyOwner {
        require(_toWhitelist != address(0), "No 0 addresses ser");
        ReferrerDetails storage ref = referrerDetails[_toWhitelist];
        require(ref.registered == true, "Ask the user to register first");
        require(_tier == 2 || _tier == 3, "Wrong tier");

        ref.isWhitelisted = _status;
        ref.discount = _discount;
        ref.rebate = _rebate;
        ref.tier = _tier;
    }

    function referKOLUnder(address _tier3, address _tier2) public onlyOwner {
        require(
            _tier3 != address(0) && _tier2 != address(0),
            "No 0 addresses ser"
        );
        ReferrerDetails storage ref = referrerDetails[_tier3];
        ReferrerDetails storage ref2 = referrerDetails[_tier2];
        require(
            ref2.registered == true && ref.registered == true,
            "Ask the user to register first"
        );
        tier3RefList[_tier3].push(_tier2);
        tier2ReferredTier3[_tier2] = _tier3;
        isTier3Referred[_tier2] = true;
    }
```

Solution

It is recommended that the event is recorded when the sensitive parameters of the contract are changed to facilitate subsequent self-audit or community review.

Status

Acknowledged

[N2] [Medium] Lack of judgment for newTP

Category: Design Logic Audit

Content

In the NarwhalTrading contract, the user can call the updateTp function to update the take profit for a given position. However, the incoming newTp is not judged here. Normally a long position should be judged to have a newTp greater than the current price, while a short position should have the opposite. If there is a lack of judgment here it may affect the user's normal trade.

Code Location: contracts/NarwhalTrading.sol#L535-563

```
    function updateTp(
        uint pairIndex,
        uint index,
        uint newTp
    ) external notContract notDone {
        address sender = _msgSender();

        StorageInterface.Trade memory t = storageT.openTrades(
            sender,
            pairIndex,
            index
        );

        StorageInterface.TradeInfo memory i = storageT.openTradesInfo(
            sender,
            pairIndex,
            index
        );

        require(t.leverage > 0, "NO_TRADE");
        require(
            block.number - i.tpLastUpdated >= limitOrdersTimelock,
            "LIMIT_TIMELOCK"
        );

        storageT.updateTp(sender, pairIndex, index, newTp);

        emit TpUpdated(sender, pairIndex, index, newTp);
    }
```

Solution

It is recommended to add a corresponding check to the newTp parameter passed in.

Status

Fixed

[N3] [Suggestion] function to repeat the effect

Category: Gas Optimization Audit

Content

In the Vester contract, the getTotalVested function is used to obtain the user's total vesting amount, and the function of the getVestedAmount function is exactly the same, which is redundant.

Code location: contracts/Tokens/Vester.sol

```
    function getTotalVested(address _account) public view returns (uint256) {
        return balances[_account].add(cumulativeClaimAmounts[_account]);
    }

    function getVestedAmount(address _account) public view returns (uint256) {
        uint256 balance = balances[_account];
        uint256 cumulativeClaimAmount = cumulativeClaimAmounts[_account];
        return balance.add(cumulativeClaimAmount);
    }
```

Solution

For functions with similar functions, it is recommended to keep only one to save gas.

Status

Fixed

[N4] [Suggestion] Redundant SafeMath is used

Category: Gas Optimization Audit

Content

The version of Solidity used in the Vester contract is version 0.8.15, which internally performs overflow checks on mathematical operations. However, the SafeMath library is used in the contract to prevent overflow, which will cause additional gas consumption.

Code location: contracts/Tokens/Vester.sol

```
pragma solidity 0.8.15;
import "@openzeppelin/contracts/utils/math/SafeMath.sol";

contract Vester is IERC20, ReentrancyGuard, Governable {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;
    ...
}
```

Solution

It is recommended to remove the unnecessary SafeMath library in Solidity 0.8.0 and above.

Status

Acknowledged;

[N5] [High] Potential incorrect decimal record issue

Category: Arithmetic Accuracy Deviation Vulnerability

Content

In the NarwhalPool contract, users can stake the tokens in the whitelist through the stake function. It will update the totalDeposited and stakedAmounts parameters according to the amount of user deposits. These two parameters will not distinguish different tokens, so all token deposits with different decimals will be recorded together. If the decimal of the tokens staked by the user is different, totalDeposited and stakedAmounts records will be wrong, which will eventually lead to errors in the calculation of rewards in the protocol.

At present, it seems that only deposits of NAR and esNAR tokens are supported, but you should be vigilant in subsequent operations.

Code location: contracts/NarwhalPool.sol

```
    function stake(uint amount, address _tokenAddress) public nonReentrant
updateReward(msg.sender) {
        ...

        totalDeposited += amount;
        totalDepositedForToken[_tokenAddress] += amount;
        stakedAmounts[msg.sender] += amount;
        ...
    }
```

Solution

It is recommended that different tokens be handled uniformly in decimal when performing stake operations.

Status

Fixed; After communicating with the project team, the project team stated that it will ensure that only esNAR and NAR tokens are staked.

[N6] [Critical] USDT Token Transfer Compatibility Issue

Category: Others

Content

In the protocol, the TokenInterface interface is used in the USDT transfer/transferFrom operation, but the transfer/transferFrom of the USDT token in the Ethereum mainnet has no return value, which will cause compatibility issues and cause the protocol to fail to perform normal USDT transfer operations.

Code location: contracts/NarwhalPool.sol

```
interface TokenInterface {
    function transfer(address, uint256) external returns (bool);
    function transferFrom(address, address, uint256) external returns (bool);
}
contract NarwhalPool is ReentrancyGuard {
    ...
    TokenInterface public USDT;
    ...
    function increaseAccTokens(uint256 _amount) external {
        require(allowedContracts[msg.sender], "ONLY_ALLOWED_CONTRACTS");
        storageT.USDT().transferFrom(msg.sender, address(this), _amount);
        ...
    }

    function harvest(bool _compound) public updateReward(msg.sender) {
        ...
            } else {
                USDT.transfer(msg.sender, pendingUSDTTotal);
            }
        } else {
            console.log("no pending reward USDT", pendingUSDTTotal);
        }
    }
}
```

Solution

It is recommended to use OpenZeppelin's SafeERC20 library for USDT transfer operations.

Status

Fixed;

[N7] [Suggestion] Redundant pause function

Category: Gas Optimization Audit

Content

In the NarwhalTrading contract, the notDone modifier is used by all functions in the contract to determine whether the contract is suspended or not, while isPaused is not used. It will cause additional gas consumption.

Code Location: contracts/NarwhalTrading.sol

```
bool public isPaused; // Prevent opening new trades
...
function pause() external onlyGov {
    isPaused = !isPaused;

    emit Paused(isPaused);
}
```

Solution

It is recommended to remove the unnecessary pause function and isPaused variable.

Status

Fixed;

[N8] [Suggestion] Incorrect use of nonReentrant modifier

Category: Gas Optimization Audit

Content

In the Vester contract, the nonReentrant modifier is used to prevent reentrant. But without any external calls in the setTransferredAverageStakedAmounts and settransferredCumulativeRewards functions, the nonReentrant modifier is used, which is redundant.

Code location: contracts/Tokens/Vester.sol

```
function setTransferredAverageStakedAmounts(
    address _account,
    uint256 _amount
) external nonReentrant {
    _validateHandler();
    transferredAverageStakedAmounts[_account] = _amount;
}

function settransferredCumulativeRewards(
    address _account,
    uint256 _amount
) external nonReentrant {
    _validateHandler();
    transferredCumulativeRewards[_account] = _amount;
}
```

Solution

It is recommended to remove unnecessary nonReentrant decorators to save gas.

Status

Fixed

[N9] [Medium] Potential non-full vesting of tokens

Category: Design Logic Audit

Content

In the VestingSchedule contract, the _getNextClaimableAmount function is used to calculate the reward to be vested. It is calculated by `balance * timeDiff / groupVestingDuration`, and in the _updateVesting function, the user's balance will be burned every time the reward is vested. Therefore, regardless of whether the user's lock-in time has exceeded the groupVestingDuration period, the user's balance will always be divided to calculate the attribution, which will cause the user's balance to never be 0, unless the user waits until the groupVestingDuration is over before claiming all belong.

Code location: contracts/Tokens/VestingSchedule.sol

```
    function _getNextClaimableAmount(
        address _account
    ) private view returns (uint256) {
        uint256 timeDiff = block.timestamp.sub(lastVestingTimes[_account]);

        uint256 balance = balances[_account];
        if (balance == 0) {
            return 0;
        }

        uint256 claimableAmount = balance.mul(timeDiff).div(
            groupVestingDuration[addressGroup[_account]]
        );
        ...
    }
```

Solution

If it is not expected, the design suggests that token vesting should be completed when the current time is greater than `timeCheck + groupVestingDuration`.

Status

Fixed

[N10] [Low] Risk of low-cost rewards through front-running

Category: Design Logic Audit

Content

In the NarwhalPool contract, the USDT reward comes from the update of the accUSDTPernarToken variable. Whenever TradingStorage deposits USDT into the contract through the increaseAccTokens function, the staked user can receive the USDT reward. In order to reduce the liquidity locking cost of long-term stakes, malicious users can stake a large amount of funds into NarwhalPool before the increaseAccTokens function is called, wait for accUSDTPernarToken to be updated and receive rewards through harvest before withdrawing funds.

Code location: contracts/NarwhalPool.sol

```
function increaseAccTokens(uint256 _amount) external {
    require(allowedContracts[msg.sender], "ONLY_ALLOWED_CONTRACTS");
    storageT.USDT().transferFrom(msg.sender, address(this), _amount);

    if (totalDeposited > 0) {
        accUSDTPernarToken += (_amount * 1e18) / totalDeposited;
        totalUSDTRewardsIncrement += _amount;
    }
}

function pendingRewardUSDTNARStake(
    address _account
) public view returns (uint) {
    if (totalDeposited == 0) {
        return 0;
    }
    UserNwxRecords storage unar = usernarrecords[_account];
    uint256 stakedToken = userTotalBalance(_account);
    uint256 pendings = (stakedToken * accUSDTPernarToken) /
        1e18 -
        unar.narDebtUSDT;
    return (pendings);
}
```

Solution

It is recommended that the duration of the user's stake be taken into account when calculating USDT rewards.

Status

Fixed

[N11] [Suggestion] Missing return value check

Category: Others

Content

The return value is not checked when calling the transfer, transferFrom and other functions of the external token contract. If the external token does not adopt the standard of EIP20, it may lead to the problem of "False Top-up"

Solution: It is recommended that when calling the function of the external token contract, if the function has a return value, it needs to check the result of the return value. SafeERC20 can be used to implement the check.

Code location: narwhal-contracts-QA/contracts/NarwhalPool.sol

```
function increaseAccTokens(uint256 _amount) external {
        require(allowedContracts[msg.sender], "ONLY_ALLOWED_CONTRACTS");
        storageT.USDT().transferFrom(msg.sender, address(this), _amount);

        if (totalDeposited > 0) {
            accUSDTPernarToken += (_amount * 1e18) / totalDeposited;
            totalUSDTRewardsIncrement += _amount;
        }
    }
```

Code location: narwhal-contracts-QA/contracts/NarwhalPool.sol

```
function harvest(bool _compound) public updateReward(msg.sender) {
        UserNwxRecords storage unar = usernarrecords[msg.sender];
        uint256 narRewards;
        uint256 USDTRewardsNAR;
       ...
            if (!_compound) {
                esToken.transfer(msg.sender, pendingTokens);
        ...
        if (pendingUSDTTotal > 0) {
            if (_compound) {
                USDT.approve(TradingVault, pendingUSDTTotal);
                ITradingVault(TradingVault).deposit(
                    pendingUSDTTotal,
                    msg.sender
                );
            } else {
                USDT.transfer(msg.sender, pendingUSDTTotal);
            }
        } else {
            console.log("no pending reward USDT", pendingUSDTTotal);
        }
    }
```

Code location: narwhal-contracts-QA/contracts/NarwhalPool.sol

```
function stake(uint amount, address _tokenAddress) public nonReentrant
updateReward(msg.sender) {
        require(isTokenWhitelisted[_tokenAddress], "Token not whitelisted");
        UserNwxRecords storage unar = usernarrecords[msg.sender];
        console.log("Before harvest");
        harvest(false);
        console.log("Passed harvest");
        IERC20(address(_tokenAddress)).transferFrom(
            msg.sender,
            address(this),
            amount
        );

        totalDeposited += amount;
        totalDepositedForToken[_tokenAddress] += amount;
        stakedAmounts[msg.sender] += amount;

        uint256 stakedToken = userTotalBalance(msg.sender);
        unar.narDebtUSDT = (stakedToken * accUSDTPernarToken) / 1e18;

        depositBalances[msg.sender][_tokenAddress] += amount;
    }
```

Code location: narwhal-contracts-QA/contracts/NarwhalPool.sol

```
    function unstake(uint amount, address _tokenAddress) public nonReentrant
updateReward(msg.sender) {
       …
```

```
        IERC20(_tokenAddress).transfer(msg.sender, am);
    }
```

Code location: narwhal-contracts-QA/contracts/TradingStorage.sol

```
    function transferUSDT(
        address _from,
        address _to,
        uint _amount
    ) external onlyTrading {
        console.log("Begin transfer USDT");
...
            TokenInterface(USDT).transfer(_to, _amount);

        } else {
            console.log("TransferFrom trading Storage");
            TokenInterface(USDT).transferFrom(_from, _to, _amount);
...
            );
        }
    }
```

Code location: narwhal-contracts-QA/contracts/TradingVaultV2.sol

```
function harvest(address _user) public updateReward(msg.sender) {
        if (balance() == 0) {
            return;
        }

        ...

        if (pendingTokens > 0) {
         ...
            cumulativeRewards[user] = nextCumulativeReward;
            esnar.transfer(user, pendingTokens);
            rewardsToken += pendingTokens;
        }
        console.log("totalSupply", totalSupply());
    }
```

Code location: narwhal-contracts-QA/contracts/TradingVaultV2.sol

```
 function distributeRewardUSDT(
        uint _amount,
        bool _send
    ) public onlyCallbacks {
        if (_send) {
            storageT.USDT().transferFrom(msg.sender, address(this), _amount);
        }
        console.log("USDT rewards distributed");
        currentBalanceUSDT = currentBalanceUSDT.add(_amount);
        totalRewardsDistributed += _amount;
    }
```

Solution

It is recommended that when calling the function of the external token contract, if the function has a return value, it needs to check the result of the return value. SafeERC20 can be used to implement the check.

Status

Fixed

[N12] [Low] Potential economic risks of the Vesting lock model

Category: Design Logic Audit

Content

In the NarwhalPool and NarwhalTradingVault contracts, users stake USDT, NAR, esNAR tokens and other whitelisted tokens to obtain esNAR token rewards, and users can deposit esNAR into the Vester contract to obtain NAR token rewards. In this model, the tokens are mutually staked and mutually rewarded, so if the market price of one of the tokens is much lower than that of other tokens, the cost of the user's stake will be reduced, and the generation of other tokens will be accelerated, resulting in NAR and esNAR The price spirals downward.

Solution

We recommend careful handling of the vesting rate of tokens and increasing the liquidity depth of protocol tokens in the market.

Status

Fixed; After communicating with the project team, the project team indicated that they will use the inPrivateTransferMode function, to allow users to stake only, prevent transfers to external entities, while allowing esnar transfers from staking contract/vester to users

[N13] [Suggestion] Redundant unstake logic issue

Category: Others

Content

In the NarwhalPool contract, users can withdraw staked tokens through the unstake function, which will perform different checking logics depending on whether the user is locked or not. But actually checks on user's depositBalances and availToWithdraw are always necessary. So checking by if-else logic based on the lockup state is redundant.

Code location: contracts/NarwhalPool.sol

```
    function unstake(uint amount, address _tokenAddress) public nonReentrant
updateReward(msg.sender) {
        require(isTokenWhitelisted[_tokenAddress], "Token not whitelisted");
        require(
            amount <= depositBalances[msg.sender][_tokenAddress],
            "AMOUNT_TOO_BIG"
        );
        UserNwxRecords storage unar = usernarrecords[msg.sender];
        harvest(false);
        uint256 am;
        (uint256 availToWithdraw, bool lockup) = userAvailToWithdraw(
            msg.sender,
            _tokenAddress
        );
        if (lockup) {
            require(amount <= availToWithdraw, "Amount too high");
            require(availToWithdraw != 0, "Nothing to withdraw");
            am = amount;
        } else {
            require(
                amount <= depositBalances[msg.sender][_tokenAddress],
                "Amount too high"
            );
            require(availToWithdraw != 0, "Nothing to withdraw");
            am = amount;


        }
        ...
    }
```

Solution

It is recommended to remove the if-else check logic for the lockup state, and keep the checks for availToWithdraw, depositBalances. And the assignment of am.

Status

Fixed

[N14] [Suggestion] storageT cannot be changed

Category: Others

Content

The following contract storageT cannot be modified, but the storageT variable of PairsStorage can be modified. If only PairsStorage changes the storageT variable, the normal operation of the project will be affected.

- narwhal-contracts-QA/contracts/LimitOrdersStorage.sol
- narwhal-contracts-QA/contracts/NarwhalPool.sol
- narwhal-contracts-QA/contracts/NarwhalPriceAggregator.sol
- narwhal-contracts-QA/contracts/NarwhalReferrals.sol
- narwhal-contracts-QA/contracts/NarwhalTrading.sol
- narwhal-contracts-QA/contracts/NarwhalTradingCallbacks.sol
- narwhal-contracts-QA/contracts/PairInfos.sol
- narwhal-contracts-QA/contracts/TradingVaultV2.sol

Code location: narwhal-contracts-QA/contracts/PairsStorage.sol

```
function changeStorageInterface(address _storage) public {
       require(msg.sender == owner);
       storageT = StorageInterface(_storage);
   }
```

Solution

It is recommended that if the project needs to modify storageT, the code should reserve the function of changeStorageInterface in other contracts to avoid that only a single contract can be modified and the project cannot run properly.

Status

Acknowledged; After communicating with the project team, the project team indicated that it will use the proxy upgradeable model. At present, the project team has added a proxy module to the TradingStorage contract, which partially solves this issue.

[N15] [Low] Authority transfer enhancement

Category: Others

Content

The govFund does not adopt the pending and access processes. If the govFund is incorrectly set, the govFund permission will be lost.

Code location: narwhal-contracts-QA/contracts/NarwhalPool.sol

```
function setGovFund(address _gov) external onlyGov {
     require(_gov != address(0), "ADDRESS_0");
     govFund = _gov;
     emit AddressUpdated("govFund", _gov);
 }

 modifier onlyGov() {
     require(msg.sender == govFund, "GOV_ONLY");
     _;
 }
```

Solution

It is recommended to adopt the pending and access processes. Only the new govFund accepts the permissions to transfer.

Status

Fixed

[N16] [High] Precision compatibility issues

Category: Arithmetic Accuracy Deviation Vulnerability

Content

The calculation of the exponent is hard-coded(1e10), rather than according to the formula in the Pyth NetWork document. When the exponent of a currency is not 8, ConvertedNumber will get an incorrect value.

Code location: narwhal-contracts-QA/contracts/NarwhalPriceAggregator.sol

```
    function tokenPriceUSDT() public view returns (uint256) {
        console.log("checking price usdt");
        PythStructs.PriceFeed memory priceFeed =
IPythTestnet(PythOracle).queryPriceFeed(USDTFeed);
        uint256 convertedNumber = uint256(uint64(priceFeed.price.price)).mul(1e10);
        console.log("convertedNumber", convertedNumber);
        return convertedNumber;
    }
```

Code location: narwhal-contracts-QA/contracts/NarwhalPriceAggregator.sol

```
…
uint256 constant PRECISION = 1e10;
…
function getPrice(
        OrderType orderType,
        bytes[] calldata updateData,
        StorageInterface.Trade memory t
    ) public payable onlyTrading returns (uint, uint256) {
       …
        IPythTestnet(PythOracle).updatePriceFeeds{
            value: getPythFee(updateData)
        }(updateData);

        PythStructs.Price memory priceP = IPythTestnet(PythOracle)
            .getPriceNoOlderThan(f.feed1,age);
      …

        uint256 convertedNumber = uint256(uint64(priceP.price));
        console.log("Price", convertedNumber);
        fulfill(orderId, convertedNumber.mul(PRECISION));
        return (orderId, convertedNumber.mul(PRECISION));
    }
```

For example, the exponent of AAPL/USD is -5, using 1e10 conversion will yield an incorrect value.

(Equity.US.AAPL/USD price feed ID：

0x49f6b65cb1de6b10eaf75e7c03ca029c306d0357e91b5311b175084a5ad55688)

Reference: https://docs.pyth.network/pythnet-price-feeds/best-practices

Solution

It is recommended to obtain the expo value in the price feed ID for calculation, instead of using hard coding.

Status

Fixed

[N17] [Suggestion] Business logic is unclear

Category: Others

Content

The fulfill function is only used in the getPrice function, so fulfill can be adjusted to internal, and the fulfill function has code comments. The annotated code needs to be confirmed whether it will affect the specific business logic, and there is an issue of unclear business logic.

Code location: /narwhal-contracts-QA/contracts/NarwhalPriceAggregator.sol

```
function fulfill(uint256 orderId, uint256 price) public {
        //recordChainlinkFulfillment(requestId)
        //uint orderId = orderIdByRequest[requestId];

        Order memory r = orders[orderId];

        //delete orderIdByRequest[requestId];
        if (!r.initiated) {
            return;
        }

        uint[] storage answers = ordersAnswers[orderId];
        answers.push(price);

        CallbacksInterface.AggregatorAnswer memory a;
        a.orderId = orderId;
        a.price = price;
        a.spreadP = PairsStorageInterface(pairsStorage).pairSpreadP(
            r.pairIndex
        );

        CallbacksInterface c = CallbacksInterface(storageT.callbacks());
        if (r.orderType == OrderType.MARKET_OPEN) {
            console.log("Market open");
            c.openTradeMarketCallback(a);
        } else if (r.orderType == OrderType.MARKET_CLOSE) {
            console.log("Market close");
            c.closeTradeMarketCallback(a);
        } else if (r.orderType == OrderType.LIMIT_OPEN) {
            console.log("Limit Open");
            c.executeOpenOrderCallback(a);
        } else if (r.orderType == OrderType.LIMIT_CLOSE) {
            console.log("Limit Close");
            c.executeCloseOrderCallback(a);
        } else {
            console.log("Update SL");
            c.updateSlCallback(a);
        }

        console.log("End callback");
        delete orders[orderId];
        delete ordersAnswers[orderId];

        // emit PriceReceived(
        //     requestId,
        //     orderId,
        //     msg.sender,
        //     r.pairIndex,
        //     price,
        //     feedPrice,
        //     r.linkFeePerNode
        // );
    }
```

The following functions are not found to be used in the contract. The project team need to confirm whether it is redundant code.

Code location: /narwhal-contracts-QA/contracts/NarwhalPriceAggregator.sol

```
  function swap(uint[] memory array, uint i, uint j) private pure {
        (array[i], array[j]) = (array[j], array[i]);
    }

    function sort(uint[] memory array, uint begin, uint end) private pure {
        if (begin >= end) {
            return;
        }

        uint j = begin;
        uint pivot = array[j];
        for (uint i = begin + 1; i < end; ++i) {
            if (array[i] < pivot) {
                swap(array, i, ++j);
            }
        }

        swap(array, begin, j);
        sort(array, begin, j);
        sort(array, j + 1, end);
    }

    function median(uint[] memory array) private pure returns (uint) {
        sort(array, 0, array.length);

        return
            array.length % 2 == 0
                ? (array[array.length / 2 - 1] + array[array.length / 2]) / 2
                : array[array.length / 2];
    }
```

Solution

It is recommended to delete unnecessary code, change the fulfill function to internal, and clarify the implementation logic of the function.

Status

Fixed; After communicating with the project team, the project team indicated that these are redundant comments.

[N18] [Low] Lack of same bots judgment

Category: Design Logic Audit

Content

It is not judged whether t.first and t.sameBlock are the same bot, this would result in a keeper getting all the rewards, weakening its expectation of avoiding gas competition.

Code location: /narwhal-contracts-QA/contracts/LimitOrdersStorage.sol

```
function storeFirstToTrigger(
       TriggeredLimitId calldata _id,
       address _bot
   ) external onlyTrading {
       TriggeredLimit storage t = triggeredLimits[_id.trader][_id.pairIndex][
           _id.index
       ][_id.order];
       t.first = _bot;
       delete t.sameBlock;
       t.block = block.number;

       emit TriggeredFirst(_id, _bot);
   }

   function storeTriggerSameBlock(
       TriggeredLimitId calldata _id,
       address _bot
   ) external onlyTrading {
       TriggeredLimit storage t = triggeredLimits[_id.trader][_id.pairIndex][
           _id.index
       ][_id.order];

       require(t.block == block.number, "TOO_LATE");
       require(t.sameBlock.length < sameBlockLimit, "SAME_BLOCK_LIMIT");

       t.sameBlock.push(_bot);

       emit TriggeredSameBlock(_id, _bot);
   }
```

Solution

It is recommended to judge t.first and t.sameBlock, and the same bot is not allowed.

Status

Acknowledged; After communicating with the project team, the project team indicated that this is the expected design.

[N19] [Critical] Compatibility risk of updateReward and allowed features

Category: Design Logic Audit

Content

In the NarwhalTradingVault contract, users can obtain rewards or deposit USDT through the harvest and deposit functions respectively. Before that, the updateReward modifier will be triggered to update the user's rewards. However, allowed users can perform deposit and harvest operations for specified users, but the object of updateReward is not the specified user but msg.sender. This will result in the user being unable to perform normal reward settlement, resulting in the risk of the user getting more rewards.

Code loccation: contracts/TradingVaultV2.sol

```
    function harvest(address _user) public updateReward(msg.sender) {
        if (balance() == 0) {
            return;
        }

        address user;
        if (allowed[msg.sender]) {
            user = _user;
        } else {
            user = msg.sender;
        }

        ...
    }

    function deposit(uint _amount, address _user) public updateReward(msg.sender)
  nonReentrant {
        require(_amount > 0, "AMOUNT_0");
        address user;
        if (allowed[msg.sender]) {
            user = _user;
        } else {
            user = msg.sender;
        }

        ...
    }
```

Solution

It is recommended to change updateReward into a function, and perform the updateReward operation on the specified user according to whether the caller is an allowed user.

Status

Fixed

[N20] [Low] referrerDetails record issue

Category: Design Logic Audit

Content

When _user is the same as _referral, there will be the same situation as refFrom and user, which is not allowed in business logic.

Code location: /narwhal-contracts-QA/contracts/NarwhalReferrals.sol

```
function signUp(address _user, address _referral) public nonReentrant {
        address user;
        if (msg.sender == address(storageT)) {
            user = _user;
        } else {
            user = msg.sender;
        }

        ReferrerDetails storage ref = referrerDetails[user];

        require(ref.registered == false, "You are already registered");
        require(
            refLinkToUser[block.number] == address(0),
            "Referral Link already taken"
        );

        ref.referralLink = block.number;
        refLinkToUser[block.number] = user;
        ref.registered = true;

        if (_referral == address(0)) {
            ref.userReferredFrom = address(0);
            referral[user] = address(0);
            ref.canChangeReferralLink = true;
            ref.discount = baseReferralDiscount;
            ref.rebate = baseReferralRebate;
        } else {
            ReferrerDetails storage refFrom = referrerDetails[_referral];
            require(refFrom.registered == true, "Referrer not registered");
            ref.userReferredFrom = _referral;
            referral[user] = _referral;
            console.log("referral[user]", referral[user]);
            console.log("user", user);
            console.log("getReferral", getReferral(user));

            ref.canChangeReferralLink = false;
            refFrom.userReferralList.push(user);
            ref.discount = refFrom.discount;
            ref.rebate = baseReferralRebate;
        }
        ref.tier = 1;
    }
```

Solution

It is recommended to add a judgment at the beginning of the function, to ensure that _user and _referral cannot be equal.

Status

Fixed

[N21] [Medium] Lack of initialization parameter validation

Category: Design Logic Audit

Content

In the NarwhalTradingCallbacks contract, the registerTrade and unregisterTrade functions can distribute rewards proportionally to treasury, marketing fund, vault, and pool addresses. These reward ratio parameters are set at the contract initialization and cannot be changed after they are set. However, the initialization does not check that _USDTVaultFeeP + _lpFeeP + _projectFeeP + _marketingFeeP <= 100. If the deployer makes a mistake in setting parameters for the deployment contract, it can happen that more rewards are allocated than expected.

Code Location: contracts/NarwhalTradingCallbacks.sol#L58-81

```
    constructor(
        StorageInterface _storageT,
        LimitOrdersInterface _nftRewards,
        PairInfoInterface _pairInfos,
        NarwhalReferralInterface _referrals,
        uint _USDTVaultFeeP,
        uint _lpFeeP,
        uint _projectFeeP,
        uint256 _marketingFeeP
    ) {
        storageT = _storageT;
        nftRewards = _nftRewards;
        pairInfos = _pairInfos;
        referrals = _referrals;

        USDTVaultFeeP = _USDTVaultFeeP;
        lpFeeP = _lpFeeP;
        projectFeeP = _projectFeeP;
        marketingFeeP = _marketingFeeP;
        Treasury = msg.sender;
        MarketingFund = msg.sender;

    }
```

Solution

It is recommended to add a judgment for _USDTVaultFeeP + _lpFeeP + _projectFeeP + _marketingFeeP <= 100.

Status

Fixed

[N22] [Critical] USDT decimal compatibility issue

Category: Arithmetic Accuracy Deviation Vulnerability

Content

The protocol ecosystem revolves around USDT, and the protocol will be deployed on the Arbitrum One network. In the Arbitrum One network, the decimal of USDT is 6, but it is processed with 18 decimal by default in the protocol. This will cause compatibility issues between the protocol's bookkeeping and USDT decimal.

Below is some sample code that is affected:

Code location: contracts/PairInfos.sol

```
    function getTradePriceImpactPure(
        uint openPrice, // PRECISION
        bool long,
        uint startOpenInterest, // 1e18 (USDT)
        uint tradeOpenInterest, // 1e18 (USDT)
        uint onePercentDepth
    )
        public
        view
        returns (
            uint priceImpactP, // PRECISION (%)
            uint priceAfterImpact // PRECISION
        )
    {
        if (onePercentDepth == 0) {
            return (0, openPrice);
        }
        priceImpactP =
            ((startOpenInterest + tradeOpenInterest / 2) * PRECISION) /
            1e18 /
            onePercentDepth;

        uint priceImpact = (priceImpactP * openPrice) / PRECISION / 100;

        priceAfterImpact = long
            ? openPrice + priceImpact
            : openPrice - priceImpact;
    }
```

contracts/TradingVaultV2.sol

```
    function _lockNAR(address _account, uint256 _amount) public {
        require(msg.sender == Vester, "Not the vesting contract");
        User storage u = users[_account];
        stakedAmounts[_account] = stakedAmounts[_account].sub(_amount);
        u.amountInLockup = u.amountInLockup.add(_amount);
    }
```

Solution

It is recommended to process the decimal when depositing USDT to be compatible with the protocol.

Status

Fixed

[N23] [High] Open a position without updating the price to increase the winning rate

Category: Design Logic Audit

Content

The price of the protocol comes from the Pyth price provider. When the user operates on the position, the NarwhalPriceAggregator contract will first update the price through the updatePriceFeeds function according to the updateData parameter passed in by the user, and then obtain the price through the getPriceNoOlderThan function to ensure that when the position is operated The prices are always the latest prices in the market. But unfortunately, in PythOracle, the updatePriceFeeds call can still be successfully made with the old updateData parameter, but the prices will not be successfully updated. This will result in the price not being up to date when the user opens a position. Malicious users can take advantage of this problem, use the old price to open a position and then use the new price to close the position, so as to increase their opening win rate and exhaust the reserves in the vault.

Code location: contracts/NarwhalPriceAggregator.sol

```
function getPrice(
    OrderType orderType,
    bytes[] calldata updateData,
    StorageInterface.Trade memory t
) public payable onlyTrading returns (uint, uint256) {
    ...
    IPythTestnet(PythOracle).updatePriceFeeds{
        value: getPythFee(updateData)
    }(updateData);

    PythStructs.Price memory priceP = IPythTestnet(PythOracle)
        .getPriceNoOlderThan(f.feed1,age);

    ...
}
```

Solution

If possible, we recommend prohibiting users from continuously opening and closing positions in the same block to prevent malicious users from taking advantage of this issue to profit.

Status

Fixed; After communicating with the project team, the project team said that by using orderExecutionTimeLimit to restrict users from operating the same order in the same block to alleviate this problem.

[N24] [Critical] Incorrect rewards account acquisition

Category: Design Logic Audit

Content

In the NarwhalTradingVault contract, users can obtain rewards through the harvest function, which distributes rewards based on the user's rewards records. However, the allowed user can refer other users to perform harvest operations, but the obtained rewards account is msg.sender, which will cause the rewards of the allowed user to be mistakenly issued as the rewards of the specified user.

Code location: contracts/TradingVaultV2.sol

```
function harvest(address _user) public updateReward(msg.sender) {
    if (balance() == 0) {
        return;
    }

    address user;
    if (allowed[msg.sender]) {
        user = _user;
    } else {
        user = msg.sender;
    }

    User storage u = users[user];

    uint pendingTokens = rewards[msg.sender];

    if (pendingTokens > 0) {
        rewards[msg.sender] = 0;
        ...
}
```

Solution

It is recommended that rewards for the correct account should be retrieved via `rewards[user]` .

Status

Fixed

[N25] [Medium] Potential risk of missing modifiers

Category: Others

Content

In the NarwhalTrading contract, all functions except the openTrade function have `notContract` and `notDone` modifiers. Users can open positions through the openTrade function. Since it does not have `notContract` and `notDone` modifiers, any user can perform openTrade operations when the contract is suspended. And if the contract user has opened a position, but because there is a `notContract` modifier in the closing position function, this will cause the contract account to never be able to close the position.

Code location: contracts/NarwhalTrading.sol

```
    function openTrade(
        StorageInterface.Trade memory t,
        LimitOrdersInterface.OpenLimitOrderType orderType, // LEGACY => market
        uint spreadReductionId,
        uint slippageP, // for market orders only
        bytes[] calldata updateData
    ) external {
    ...
}
```

Solution

It is recommended to add `notContract` and `notDone` modifiers to the openTrade function.

Status

Fixed

[N26] [Low] Risk of not being able to lock

Category: Design Logic Audit

Content

In the Vester contract, When the user makes a deposit, The _deposit function will calculate the reward based on the amount of collateral the user has in the pool and call the _lockNAR function of the pool contract to lock some of the collateral (the amount locked is calculated based on the reward).

However, if the user is not harvested in the pool, the cumulativeReward of the user is zero. In this case, if the handle role does not set the user's transferredAverageStakedAmounts, this will result in a zero result from the getPairAmounth function after the calculation. Thus, the _lockNAR function cannot be called properly to lock the partial collateral.

Code location: contracts/Tokens/Vester.sol

```
    function _deposit(
        address _account,
        uint256 _amount
    ) private returns (uint256) {
        ...

        uint256 pairAmountDiff;

        uint256 pairAmount = pairAmounts[_account];
        uint256 nextPairAmount = getPairAmount(balances[_account], _account);
        console.log("nextPairAmount", nextPairAmount);
        console.log("pairAmount", pairAmount);

        if (nextPairAmount > pairAmount) {
            pairAmountDiff = nextPairAmount.sub(pairAmount);
            console.log("pairAmountDiff", pairAmountDiff);

            uint256 totalStaked = IPoolRewards(NarwhalPool).getUserTokenBalance(
                _account
            );
            console.log("passed pool rewards call");
            require(
                totalStaked >= pairAmountDiff,
                "Not enough balance locked up in pool"
            );
            IPoolRewards(NarwhalPool)._lockNAR(_account, pairAmountDiff);
            _mintPair(_account, pairAmountDiff);
        }

        ...
    }
```

Solution

It is recommended to first help the user to carry out the harvest operation when depositing or ensure that the user's transferredAverageStakedAmounts have been set by the handle role before depositing.

Status

Acknowledged; After communicating with the project team, the project team indicated that the harvest check will be performed indirectly through hasMaxVestableAmount.

[N27] [Suggestion] Minimum USDT position check issue

Category: Design Logic Audit

Content

In the openTrade function of the NarwhalTrading contract, it will check whether the user's opening position is less than maxPosUSDT, but does not check its minimum value, but only checks that the leverage position must be greater than pairMinLevPosUSDT. This will result in users still being able to open positions using smaller positions with greater leverage. In other words, due to the pairMinLevPosUSDT check, the user's opening position cost is further reduced. When these small positions require the keeper to operate, the keeper may choose to ignore these small positions because the cost is greater than the benefit. This can cause the protocol to accumulate many bad positions.

Code location: contracts/NarwhalTrading.sol

```
    function openTrade(
        StorageInterface.Trade memory t,
        LimitOrdersInterface.OpenLimitOrderType orderType, // LEGACY => market
        uint spreadReductionId,
        uint slippageP, // for market orders only
        bytes[] calldata updateData
    ) external {
        ...
        require(t.positionSizeUSDT <= maxPosUSDT, "ABOVE_MAX_POS");
        require(
            t.positionSizeUSDT * t.leverage >=
                pairsStored.pairMinLevPosUSDT(t.pairIndex),
            "BELOW_MIN_POS"
        );
        ...
    }
```

Solution

It is recommended to add the mimPosUSDT variable for minimum position checks.

Status

Acknowledged; After communicating with the project team, the project team indicated that this is the expected design.

[N28] [High] Missing check for maximum PnL when updating Take Profit

Category: Design Logic Audit

Content

In the NarwhalTrading contract, users can update the take-profit price through the updateTp function. There is MAX_GAIN_P in the protocol, which requires that the user's maximum profit cannot exceed 900%, but the MAX_GAIN_P check is not performed on the user's updated newTp in the updateTp function. This will allow users to bypass the maximum profit limit of 900% through the updateTp function.

Code location: contracts/NarwhalTrading.sol

```
    function updateTp(
        uint pairIndex,
        uint index,
        uint newTp
    ) external notContract notDone {
        ...
    }
```

Solution

It is recommended to perform MAX_GAIN_P check on the user's newTp when performing an updateTp operation.

Status

Fixed

[N29] [Suggestion] Redundant temp variable in closeTradeMarket

Category: Others

Content

In the closeTradeMarket function of the NarwhalTrading contract, set tempSlippage and tempSpreadReduction to 0 after the getPrice operation is completed. But actually there is no need to use these temporary variables when closing a position.

Code location: contracts/NarwhalTrading.sol

```
    function closeTradeMarket(
        uint pairIndex,
        uint index,
        bytes[] calldata updateData
    ) external notContract notDone {
        ...
        tempSlippage = 0;
        tempSpreadReduction = 0;
        ...
    }
```

Solution

It is recommended to remove redundant tempSlippage and tempSpreadReduction parameters.

Status

Fixed

[N30] [Suggestion] `getPrice` missing check for `UPDATE_SL` type

Category: Others

Content

In the getPrice function of the NarwhalPriceAggregator contract, different operations are performed according to the OrderType, but it does not handle the `UPDATE_SL` type separately, which will cause the storePendingSlOrder operation to be executed when performing LIMIT_OPEN, LIMIT_CLOSE, and UPDATE_SL.

Code location: contracts/NarwhalPriceAggregator.sol

```
    function getPrice(
        OrderType orderType,
        bytes[] calldata updateData,
        StorageInterface.Trade memory t
    ) public payable onlyTrading returns (uint, uint256) {
        ...
        if (orderType == OrderType.MARKET_CLOSE) {
            ...
        } else if (orderType == OrderType.MARKET_OPEN) {
            ...
        } else {
            storePendingSlOrder(
                orderId,
                PendingSl(
                    t.trader,
                    t.pairIndex,
                    t.index,
                    t.openPrice,
                    t.buy,
                    INarwhal(NarwhalTrading).tempSL()
                )
            );
        }
        ...
    }
```

Solution

It is recommended to use `else if` to judge UPDATE_SL.

Status

Acknowledged

[N31] [Medium] Risk of excessive permissions

Category: Authority Control Vulnerability Audit

Content

In the protocol, certain privileged roles are designated as Gov roles, and the Gov roles can set the handler role, oracle machine, and sensitive parameters in the agreement. However, the Governance module was not included in the scope of this audit, so we cannot predict whether privileged roles will be managed as expected.

Here are some scenarios where privileged roles have too much authority:

In BaseToken, the handler role can directly transfer tokens from other user accounts through the transferFrom function. The handler role is set by the Gov role.

In the MintableBaseToken contract, the Minter role can burn the tokens of any user, and the Minter role is set by the Gov role.

In the Vester contract, the handler role can claim the rewards of any user through the claimForAccount function and transfer them to the specified address. The handler role is set by the Gov role.

In the Trading module, the Gov role can modify various sensitive parameters, which involve the transfer of tokens, the modification of the storage contract, the modification of the oracle machine, etc.

Solution

In the early stage of the operation of the protocol, in order to ensure the timely response to various emergencies and the rapid iteration of the protocol, it is recommended that the Gov role be managed by a multisign to avoid single-point risks.

But in the long run, the role of Gov should be handed over to community governance after the protocol is running stably, so as to avoid the risk of power concentration.

Status

Acknowledged; After communicating with the project team, the project team stated that it will use timelock to mitigate this risk. However, the protocol has not yet been deployed on the chain, and the ownership of Gov has not been transferred, so risks still exist.

[N32] [Low] Time check inconsistency

Category: Design Logic Audit

Content

In the VestingSchedule contract, when the user calls the claim function to get tokens, it will first determine the current timestamp block.timestamp >= timeCheck (startTime.add(groupCliff[addressGroup[msg.sender]])), followed by will call the _updateVesting function and calculate the timeDiff internally by subtracting lastVestingTimes[_account] from block.timestamp.

There is a risk that when the owner calls the start function first and then calls addAmountToInvestors some time later, the lastVestingTimes[_accounts[i]] may be larger than startTime.add(groupCliff[addressGroup[msg.sender]]. And when startTime.add(groupCliff[addressGroup[msg.sender]]) <= block.timestamp <= lastVestingTimes[_accounts[i]], this will cause an error in the claim operation.

Code Location: contracts/Tokens/VestingSchedule.sol

```
    function claim() external nonReentrant returns (uint256) {
        uint256 timeCheck = startTime.add(groupCliff[addressGroup[msg.sender]]);
        require(block.timestamp >= timeCheck,"Not time to claim yet");
        require(started == true, "Not started");
        require(allowedClaimers[msg.sender] == true, "Not allowed to claim");
        return _claim(msg.sender, msg.sender);
    }

    function _claim(
        address _account,
        address _receiver
    ) private returns (uint256) {
        _updateVesting(_account);
        ...
    }

    function _updateVesting(address _account) private {
        uint256 amount = _getNextClaimableAmount(_account);
        lastVestingTimes[_account] = block.timestamp;
        ...
    }

    function _getNextClaimableAmount(
        address _account
    ) private view returns (uint256) {
        uint256 timeDiff = block.timestamp.sub(lastVestingTimes[_account]);
        ...
    }
```

Solution

Make sure that the start function is called after the addAmountToInvestors function.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002304200003	SlowMist Security Team	2023.03.29 - 2023.04.20	Medium Risk

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 4 critical risks, 4 high risks, 5 medium risks, 7 low risks, and 12 suggestions. All the findings were fixed or acknowledged. The code was not deployed to the mainnet. Since the protocol has not yet been deployed on the chain, and the ownership of Gov has not been transferred, risks still exist.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

SLOWMIST

Official Website

www.slowmist.com

✉

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist