Smart contract security audit report

Mensa smart contract security audit report

Audit Team : Noneage security team

Audit Date : Oct 18, 2021

Mensa Smart Contract Security Audit Report

1. Overview

On Sep 22, 2021, the security team of Noneage Technology received the security audit request of the **Mensa project**. The team completed the audit of the **Mensa smart contract** on Oct 18, 2021. During the audit process, the security audit experts of Noneage Technology and the Mensa project interface Personnel communicate and maintain symmetry of information, conduct security audits under controllable operational risks, and avoid risks to project generation and operations during the testing process.

Through communicat and feedback with Mensa project party, it is confirmed that the loopholes and risks found in the audit process have been repaired or within the acceptable range. The result of this Mensa smart contract security audit: **passed**.

Audit Report MD5：FF1994A4ACCEB5EA3AA20289D17AD441

2. Background

2.1 Project Description

Project name：Mensa

Contract type：DeFi

Code language：Solidity

Source code：https://github.com/mensapro/mensa-protocol

Audit version：commit de38ee7be787fe0e4c2484fc8c497e07ab9475a1

Contract file：TokenDistributor.sol，FeeProvider.sol，stake.sol，MToken.sol，MensaManager.sol，MensaParametersProvider.sol，UintStorage.sol，AddressStorage.sol，MensaAddressesProvider.sol，GenericOracleI.sol，InterestRateOracle.sol，PriceOracle.sol，MockAggregatorUSDC.sol，MockAggregatorKNC.sol，MockAggregatorUSDT.sol，MockAggregatorMKR.sol，MockAggregatorWBTC.sol，MockAggregatorDAI.sol，MockAggregatorLINK.sol，MockAggregatorBase.sol，MockAggregatorSUSD.sol，MockAggregatorREP.sol，MockAggregatorZRX.sol，MockAggregatorMANA.sol，MockAggregatorTUSD.sol，MockAggregatorBAT.sol，MockDAI.sol，MockMANA.sol，MockKNC.sol，MockBAT.sol，MockLINK.sol，MockZRX.sol，MockUSDT.sol，MockUSDC.sol，MintableERC20.sol，

MockSUSD.sol， MockTUSD.sol， MockMKR.sol ， MockWBTC.sol ，
MockREP.sol， MockMensaCore.sol， MensaQuery.sol，
MensaMinter.sol ，MensaToken.sol， BandProxyPriceProvider.sol ，
tokenSwitch.sol， IERC20DetailedBytes.sol ，
ChainlinkProxyPriceProvider.sol， WalletBalanceProvider.sol ，
MensaLiquidationManager.sol， Mensa.sol ，MensaDataProvider.sol ，
MensaConfigurator.sol， MensaCore.sol ，IFlashLoanReceiver.sol ，
DefaultReserveInterestRateStrategy.sol，FlashLoanReceiverBase.sol

2.2 Audit Range

Mensa officially provides contract documents and documents corresponding to MD5：

TokenDistributor.sol	E732C65EF1BD52598DA4A9FD2B60ECB8
FeeProvider.sol	A54ACEA1366FF384AA146BC83E90BCE6
stake.sol	C2CA2115F1BF045FF5552102CAE82EA7
MToken.sol	43EC406B2F4EF7B9422DA0CBC1041424
MensaAddressesProvider.sol	2024F9240B3B631BB9D4F55151239067
UintStorage.sol	EFC20BC1D7808ABBD115B8BAFA1323AD
AddressStorage.sol	C12425238ADD3708B841B4B5F278268E
MensaParametersProvider.sol	4B6AAB4520B450A4A2FC6C58DF40C4CC
MensaManager.sol	E75D972699C391CA744FBAB7399A904A
GenericOracleI.sol	79ECF7E540C043149B5B5DDEF77296BF
InterestRateOracle.sol	1D0CF378C236D9F5CD8A49567118D1DF
PriceOracle.sol	EEC7B9BA2799CF07BBCC9BABB92632BC
MockAggregatorUSDC.sol	9E8ADD4DCC5D77057BDD467617DE0D11
MockAggregatorKNC.sol	C16CC947EDD407A9776D65D5E3D18C9A
MockAggregatorUSDT.sol	7F8B8DE0066750390CCEDF03FF75F693
MockAggregatorMKR.sol	4C3A3F24AFE8BA6ABED7C18AC8118DCD
MockAggregatorWBTC.sol	1E573ED3C68AE9AC413151AF1E796EB2

MockAggregatorDAI.sol F81E4FCB2DCDD6FB7BD7F4A5E022A410

MockAggregatorLINK.sol 3C4334979EE33384C4373B374562340E

MockAggregatorBase.sol B4C5C9A201AC94E7F70C2ED5A2351F5E

MockAggregatorSUSD.sol E507654C3AED98F00A1FA649D2555888

MockAggregatorREP.sol 0A449B0242E8977274D99A6242CB9FC6

MockAggregatorZRX.sol E74A7E97D4A0B4AAEB14C031E950426F

MockAggregatorMANA.sol 94AF38D49E41B460779A4D67CD38CA9E

MockAggregatorTUSD.sol CE030CD159454646FD4A3DED992B9A4B

MockAggregatorBAT.sol 5C63992FDB80D882C52BB0903739238D

MockMensaCore.sol B136E7CEA532B70A8E53733D1E9651DC

MensaMinter.sol 79BF8493CDB1166251FD210786281F3E

MensaQuery.sol 2EFE322FFCE2962CDBA43F5C66CC30E8

MensaToken.sol 2E2823F3BD3C258B7AADE4826CAA4714

BandProxyPriceProvider.sol 9719B237079F39BA4A9770096CD20414

tokenSwitch.sol 73DA0BA2915756ABA0770E265E12C3FD

IERC20DetailedBytes.sol 602BA30C825913860BAA77B509877719

ChainlinkProxyPriceProvider.sol 4BB9C5F3F4D4BA35C54F4C9690BE67B5

WalletBalanceProvider.sol E8D5FF28060D16A41DC0EAF81E6B88AB

MensaLiquidationManager.sol A19B4FFE646F30EF3ACFF023306DEE5B

Mensa.sol 96F3DACAE295B266FE9C1658871A9BD4

MensaDataProvider.sol 896B241C5F83FF41967BD261C30111E0

MensaConfigurator.sol 80439B15563D956C7AE21568E94A7D49

MensaCore.sol A33A2DA6841D3217439008117B14B2BC

FlashLoanReceiverBase.sol 3C22D0464CCEE55C7DC5A11017087F47

DefaultReserveInterestRateStrategy.sol
CDA4552EEC3BAE0B3C8DE10653B44188

3. Project contract details

3.1 Directory Structure

```
└─Mensa contracts
  ├──configuration
  │    AddressStorage.sol
  │    MensaAddressesProvider.sol
  │    MensaManager.sol
  │    MensaParametersProvider.sol
  │    UintStorage.sol
  ├──fees
  │    FeeProvider.sol
  │    stake.sol
  │    TokenDistributor.sol
  ├──flashloan
  │  ├──base
  │  │    FlashLoanReceiverBase.sol
  │  └──interfaces
  │       IFlashLoanReceiver.sol
  ├──interfaces
  │    IChainlinkAggregator.sol
  │    IFeeProvider.sol
  │    IInterestRateOracle.sol
  │    IKyberNetworkProxyInterface.sol
  │    IMensaAddressesProvider.sol
  │    IMensaMinter.sol
```

```
│   IPriceOracle.sol
│   IPriceOracleGetter.sol
│   IReserveInterestRateStrategy.sol
├──libraries
│   │   CoreLibrary.sol
│   │   EthAddressLib.sol
│   │   WadRayMath.sol
│   └──openzeppelin-upgradeability
│       AdminUpgradeabilityProxy.sol
│       BaseAdminUpgradeabilityProxy.sol
│       BaseUpgradeabilityProxy.sol
│       Initializable.sol
│       InitializableAdminUpgradeabilityProxy.sol
│       InitializableUpgradeabilityProxy.sol
│       Proxy.sol
│       UpgradeabilityProxy.sol
│       VersionedInitializable.sol
├──mensa
│   DefaultReserveInterestRateStrategy.sol
│   Mensa.sol
│   MensaConfigurator.sol
│   MensaCore.sol
│   MensaDataProvider.sol
│   MensaLiquidationManager.sol
├──mint
│   MensaMinter.sol
```

```
│   MensaQuery.sol
│   MensaToken.sol
├──misc
│   BandProxyPriceProvider.sol
│   ChainlinkProxyPriceProvider.sol
│   IERC20DetailedBytes.sol
│   tokenSwitch.sol
│   WalletBalanceProvider.sol
├──mocks
│   ├──oracle
│   │   │   GenericOracleI.sol
│   │   │   InterestRateOracle.sol
│   │   │   PriceOracle.sol
│   │   └──CLAggregators
│   │       MockAggregatorBase.sol
│   │       MockAggregatorBAT.sol
│   │       MockAggregatorDAI.sol
│   │       MockAggregatorKNC.sol
│   │       MockAggregatorLINK.sol
│   │       MockAggregatorMANA.sol
│   │       MockAggregatorMKR.sol
│   │       MockAggregatorREP.sol
│   │       MockAggregatorSUSD.sol
│   │       MockAggregatorTUSD.sol
│   │       MockAggregatorUSDC.sol
│   │       MockAggregatorUSDT.sol
```

```
│   │       MockAggregatorWBTC.sol
│   │       MockAggregatorZRX.sol
│   ├──tokens
│   │   MintableERC20.sol
│   │   MockBAT.sol
│   │   MockDAI.sol
│   │   MockKNC.sol
│   │   MockLINK.sol
│   │   MockMANA.sol
│   │   MockMKR.sol
│   │   MockREP.sol
│   │   MockSUSD.sol
│   │   MockTUSD.sol
│   │   MockUSDC.sol
│   │   MockUSDT.sol
│   │   MockWBTC.sol
│   │   MockZRX.sol
│   └──upgradeability
│       MockMensaCore.sol
└──tokenization
    MToken.sol
```

3.2 Contract details

TokenDistributor contract

Name	Parameter	Attributes
getDistribution	none	public
getRevision	none	internal
internalTrade	address _from uint256 _amount	internal
internalBurn	uint256 _amount	internal

FeeProvider contract

Name	Parameter	Attributes
getRevision	none	internal
initialize	address _addressesProvider	public
calculateLoanOriginationFee	address _user uint256 _amount	external
getLoanOriginationFeePercentage	none	external

MensaStaking contract

Name	Parameter	Attributes
addAsset	address _assetAddress	onlyOwner
stake	uint _MensaAmount	external
unstake	uint _MensaAmount	external
assetIncome	address _assetAddress	public
updateAssetsFeePerStake	none	internal
_updateAssetFeePerStake	address asset	internal
_computeAssetFeePerStake	address asset	public
getPendingGain	address _user address asset	external

_getPendingGain	address _user address asset	internal
_updateUserSnapshots	none	internal
_sendETHGainToUser	uint ETHGain	internal
_requireCallerIsTroveManager	none	internal
_requireCallerIsBorrowerOperations	none	internal
_requireCallerIsActivePool	none	internal
_requireUserHasStake	uint currentStake	internal
_requireNonZeroAmount	uint _amount	internal

MToken contract

Name	Parameter	Attributes
_transfer	address _from address _to uint256 _amount	internal
redirectInterestStream	address _to	external
redirectInterestStreamOf	address _from address _to	external
allowInterestRedirectionTo	address _to	external
redeem	uint256 _amount	external
mintOnDeposit	address _account uint256 _amount	onlyMensa
burnOnLiquidation	address _account uint256 _value	onlyMensaStaff
transferOnLiquidation	address _from address _to uint256 _value	onlyMensaStaff
balanceOf	address _user	public

principalBalanceOf	address _user	external
totalSupply	none	public
isTransferAllowed	address _user uint256 _amount	public
getUserIndex	address _user	external
getInterestRedirectionAddress	address _user	external
getRedirectedBalance	address _user	external
cumulateBalanceInternal	address _user	internal
updateRedirectedBalanceOfRedirectionAddressInternal	address _user uint256 _balanceToAdd uint256 _balanceToRemove	internal
calculateCumulatedBalanceInternal	address _user uint256 _balance	internal
executeTransferInternal	address _from address _to uint256 _value	internal
redirectInterestStreamInternal	address _from address _to	internal
resetDataOnZeroBalanceInternal	address _user	internal

MensaManager contract

Name	Parameter	Attributes
initAddressProvider	MensaAddressesProvider _addressesProvider	onlyOwner
getRevision	none	internal
initReserve	address _reserve uint8 _underlyingAssetDecimals address _interestRateStrategyAddress	onlyOwner

initReserveWithData	address _reserve string _mTokenName string _mTokenSymbol uint8 _underlyingAssetDecimals address _interestRateStrategyAddress	onlyOwner
removeLastAddedReserve	address _reserveToRemove	onlyOwner
enableBorrowingOnReserve	address _reserve bool _stableBorrowRateEnabled	onlyOwner
disableBorrowingOnReserve	address _reserve	onlyOwner
enableReserveAsCollateral	address _reserve uint256 _baseLTVasCollateral uint256 _liquidationThreshold uint256 _liquidationBonus	onlyOwner
disableReserveAsCollateral	address _reserve	onlyOwner
enableReserveStableBorrowRate	address _reserve	onlyOwner
disableReserveStableBorrowRate	address _reserve	onlyOwner
activateReserve	address _reserve	onlyOwner
deactivateReserve	address _reserve	onlyOwner
freezeReserve	address _reserve	onlyOwner
unfreezeReserve	address _reserve	onlyOwner
setReserveBaseLTVasCollateral	address _reserve uint256 _ltv	onlyOwner
setReserveLiquidationThreshold	address _reserve uint256 _threshold	onlyOwner
setReserveLiquidationBonus	address _reserve uint256 _bonus	onlyOwner
setReserveDecimals	address _reserve uint256 _decimals	onlyOwner

setReserveInterestRateStrategyAddress	address _reserve address _rateStrategyAddress	onlyOwner
refreshMensaCoreConfiguration	none	onlyOwner

MensaParametersProvider contract

Name	Parameter	Attributes
getRevision	none	internal
initialize	address _addressesProvider	public
getMaxStableRateBorrowSizePercent	none	external
getRebalanceDownRateDelta	none	external
getFlashLoanFeesInBips	none	external

UintStorage contract

Name	Parameter	Attributes
getUint	bytes32 _key	public
_setUint	bytes32 _key uint256 _value	internal

AddressStorage contract

Name	Parameter	Attributes
getAddress	bytes32 _key	public
_setAddress	bytes32 _key address _value	internal

MensaAddressesProvider contract

Name	Parameter	Attributes
getMensa	none	public
setMensaImpl	address _mensa	onlyOwner
getMensaCore	none	public
setMensaCoreImpl	address _mensaCore	onlyOwner
getMensaConfigurator	none	public
setMensaConfiguratorImpl	address _configurator	onlyOwner
getMensaDataProvider	none	public
setMensaDataProviderImpl	address _provider	onlyOwner
getMensaParametersProvider	none	public
setMensaParametersProviderImpl	address _parametersProvider	onlyOwner
getFeeProvider	none	public
setFeeProviderImpl	address _feeProvider	onlyOwner
getMensaLiquidationManager	none	public
setMensaLiquidationManager	address _manager	onlyOwner
getMensaManager	none	public
setMensaManager	address _mensaManager	onlyOwner
getPriceOracle	none	public
setPriceOracle	address _priceOracle	onlyOwner
getInterestRateOracle	none	public
setInterestRateOracle	address _interestRateOracle	onlyOwner
getTokenDistributor	none	public
setTokenDistributor	address _tokenDistributor	onlyOwner
getMensaMinter	none	public
setMensaMinter	address _mensaMinter	onlyOwner
updateImplInternal	bytes32 _id address _newAddress	internal

GenericOracleI contract

Name	Parameter	Attributes
getAssetPrice	address _asset	external
getEthUsdPrice	none	external

InterestRateOracle contract

Name	Parameter	Attributes
getMarketBorrowRate	address _asset	external
setMarketBorrowRate	address _asset uint256 _rate	onlyOwner
getMarketLiquidityRate	address _asset	external
setMarketLiquidityRate	address _asset uint256 _rate	onlyOwner

PriceOracle contract

Name	Parameter	Attributes
getAssetPrice	address _asset	external
setAssetPrice	address _asset uint256 _price	onlyOwner
getEthUsdPrice	none	external
setEthUsdPrice	uint256 _price	onlyOwner

MockAggregatorBase contract

Name	Parameter	Attributes
latestAnswer	none	external

MintableERC20 contract

Name	Parameter	Attributes
mint	uint256 value	public

MockMensaCore contract

Name	Parameter	Attributes
getRevision	none	internal
initialize	MensaAddressesProvider _addressesProvider	public
updateReserveInterestRatesAndTimestampInternal	address _reserve uint256 _liquidityAdded uint256 _liquidityTaken	internal

MensaQuery contract

Name	Parameter	Attributes
getGroupInfo	uint256 _pid uint256 _gid	external
pendingMensa	uint256 _pid uint256 _gid address _user	external
earnings	uint256 pid uint256 gid uint256 duringBlocks uint256 baseReservsePrice uint256 mintReservePrice	public
mintTokenPerBlock	uint256 pid uint256 gid	public
mintedToken	uint256 pid uint256 gid uint256 poolStart uint256 poolEnd	public

MensaMiner contract

Name	Parameter	Attributes
setLp	address addressLP	onlyOwner
mint	address _to uint256 _amount	internal
poolLength	none	public
createPool	uint256 _pid uint256 _poolCap uint256 _startBlock uint256 _endBlock	onlyOwner

setGroup	uint256 _pid uint256 _gid uint256 _allocPoint	onlyOwner
getMultiplier	uint256 _from uint256 _to uint256 _start uint256 _end	internal
massUpdateGroups	uint256 _pid	internal
calculateMGR	uint256 point uint256 base	internal
updateUserProtectDuration	address userAddr UserInfo u uint256 amount uint256 pending uint256 poolPrice bool unlockedOnly	internal
updateGroup	uint256 _pid uint256 _gid	internal
getPoolInfo	uint256 _pid	public
getGroupInfo	uint256 _pid uint256 _gid	public
_deposit	uint256 _pid uint256 _gid address u uint256 _amount	internal
_withdraw	uint256 _pid uint256 _gid address u uint256 _amount bool unlockedOnly	internal
pendingMensa	uint256 _pid uint256 _gid address _user	public
selectPool	none	internal
getUserAmount	uint256 _gid address _user	public
_priceRate	PoolInfo p Group g UserInfo u	internal
userGainPrice	uint256 _pid uint256 _gid address _u uint256 _amount bool isDeposit	internal
updatePricePerStake	uint256 _pid uint256 amount	internal
updateGroupStake	uint256 _pid uint256 _gid uint256 amount	internal
mintMensaToken	address _reserve address _user uint256 _gid uint256 _amount uint256 _reserveDEC uint256 _price	onlyMensa
getUserReserveAVP	uint256 _pid uint256 _gid address _reserve address _user	public
_mintMensaToken	address _user uint256 _pid uint256 _gid uint256 _amount	internal
withdrawMensaToken	address _reserve address _user uint256 _gid uint256 _amount uint256 _dec bool unlockedOnly	onlyMensa

_withdrawMensaToken	address _user uint256 _pid uint256 _gid uint256 _amount bool unlockedOnly	internal
_withdrawPendingMensaToken	uint256 _gid bool unlockedOnly	public
withdrawPendingMensaToken	bool unlockedOnly	public
setPoolInited	uint256 _pid	onlyOwner
deposit	uint256 _pid uint256 _gid address u uint256 _amount	onlyOwner
withdraw	uint256 _pid uint256 _gid address u uint256 _amount	onlyOwner

MensaToken contract

Name	Parameter	Attributes
MensaTokenMint	address _to uint256 _amount	internal
totalSupply	none	public
cap	none	public

BandOracle contract

Name	Parameter	Attributes
getReferenceData	string _base string _quote	external
symbol	none	external
getAssetPrice	address _asset	external
internalSetFallbackOracle	address _fallbackOracle	internal
setAssetSource	address _asset string _symbol	onlyOwner
unsetAssetSource	address _asset	onlyOwner
internalSetAssetsSource	address _asset string _symbol	internal
internalUnSetAssetsSource	address _asset	internal
getBasePrice	none	external
getAssetPrice	address _asset	external

tokenSwitch contract

Name	Parameter	Attributes
goSwitch	uint256 amount	public
setRate	uint256 _rate	onlyOwner
cleanup	none	onlyOwner

ChainlinkProxyPriceProvider contract

Name	Parameter	Attributes
setFallbackOracle	address _fallbackOracle	onlyOwner
internalSetFallbackOracle	address _fallbackOracle	internal
getAssetPrice	address _asset	public
getSourceOfAsset	address _asset	external
getFallbackOracle	none	external

WalletBalanceProvider contract

Name	Parameter	Attributes
balanceOf	address _user address _token	public
getUserWalletBalances	address _user	public

MensaLiquidationManager contract

Name	Parameter	Attributes
init	MensaAddressesProvider _addressesProvider	onlyOwner
getRevision	none	internal

liquidationCall	address _collateral address _reserve address _user address _caller uint256 _purchaseAmount bool _receiveMToken	external
calculateAvailableCollateralToLiquidate	address _collateral address _principal uint256 _purchaseAmount uint256 _userCollateralBalance	internal

Mensa contract

Name	Parameter	Attributes
getRevision	none	internal
initialize	MensaAddressesProvider _addressesProvider	public
deposit	address _reserve uint256 _amount uint16 _referralCode	onlyAmountGreaterThanZero
redeemUnderlying	address _reserve address _user uint256 _amount uint256 _mTokenBalanceAfterRedeem	onlyAmountGreaterThanZero
borrow	address _reserve uint256 _amount uint256 _interestRateMode uint16 _referralCode	onlyAmountGreaterThanZero
repay	address _reserve uint256 _amount address _onBehalfOf	onlyAmountGreaterThanZero

swapBorrowRateMode	address _reserve	onlyUnfreezedReserve
rebalanceStableBorrowRate	address _reserve address _user	onlyActiveReserve
setUserUseReserveAsCollateral	address _reserve bool _useAsCollateral	onlyUnfreezedReserve
liquidationCall	address _collateral address _reserve address _user address _caller uint256 _purchaseAmount bool _receiveMToken	onlyActiveReserve
flashLoan	address _receiver address _reserve uint256 _amount bytes _params	onlyAmountGreaterThanZero
getReserveConfigurationData	address _reserve	external
getReserveData	address _reserve	external
getUserAccountData	address _user	external
getUserReserveData	address _reserve address _user	external
getReserves	none	external
requireReserveActiveInternal	address _reserve	internal
requireReserveNotFreezedInternal	address _reserve	internal
requireAmountGreaterThanZeroInternal	uint256 _amount	internal

MensaDataProvider contract

Name	Parameter	Attributes
getRevision	none	internal
initialize	MensaAddressesProvider _addressesProvider	public
calculateUserGlobalData	address _user	public
balanceDecreaseAllowed	address _reserve address _user uint256 _amount	external

getHealthFactorAfterDecrease	address _reserve address _user uint256 _amount bool _isDecrease	external
calculateCollateralNeededInETH	address _reserve uint256 _amount uint256 _fee uint256 _userCurrentBorrowBalanceTH uint256 _userCurrentFeesETH uint256 _userCurrentLtv	external
calculateAvailableBorrowsETHInternal	uint256 collateralBalanceETH uint256 borrowBalanceETH uint256 totalFeesETH uint256 ltv	internal
calculateHealthFactorFromBalancesInternal	uint256 collateralBalanceETH uint256 borrowBalanceETH uint256 totalFeesETH uint256 liquidationThreshold	internal
getHealthFactorLiquidationThreshold	none	public
getReserveConfigurationData	address _reserve	external
getReserveData	address _reserve	external
getUserAccountData	address _user	external
getUserReserveData	address _reserve address _user	external
getReserveValues	address _reserve	external

MensaConfigurator contract

Name	Parameter	Attributes
getRevision	none	internal
initialize	MensaAddressesProvider _addressesProvider	public
initReserve	address _reserve uint8 _underlyingAssetDecimals address _interestRateStrategyAddress	onlyMensaManager

initReserveWithData	address _reserve string _mTokenName string _mTokenSymbol uint8 _underlyingAssetDecimals address _interestRateStrategyAddress	onlyMensaManager
removeLastAddedReserve	address _reserveToRemove	onlyMensaManager
enableBorrowingOnReserve	address _reserve bool _stableBorrowRateEnabled	onlyMensaManager
disableBorrowingOnReserve	address _reserve	onlyMensaManager
enableReserveAsCollateral	address _reserve uint256 _baseLTVasCollateral uint256 _liquidationThreshold uint256 _liquidationBonus	onlyMensaManager
disableReserveAsCollateral	address _reserve	onlyMensaManager
enableReserveStableBorrowRate	address _reserve	onlyMensaManager
disableReserveStableBorrowRate	address _reserve	onlyMensaManager
activateReserve	address _reserve	onlyMensaManager
deactivateReserve	address _reserve	onlyMensaManager
freezeReserve	address _reserve	onlyMensaManager
unfreezeReserve	address _reserve	onlyMensaManager
setReserveBaseLTVasCollateral	address _reserve uint256 _ltv	onlyMensaManager
setReserveLiquidationThreshold	address _reserve uint256 _threshold	onlyMensaManager
setReserveLiquidationBonus	address _reserve uint256 _bonus	onlyMensaManager
setReserveDecimals	address _reserve uint256 _decimals	onlyMensaManager

setReserveInterestRateStrategyAddress	address _reserve address _rateStrategyAddress	onlyMensaManager
refreshMensaCoreConfiguration	none	onlyMensaManager

MensaCore contract

Name	Parameter	Attributes
getRevision	none	internal
initialize	MensaAddressesProvider _addressesProvider	public
updateStateOnDeposit	address _reserve address _user uint256 _amount bool _isFirstDeposit	onlyMensa
updateStateOnRedeem	address _reserve address _user uint256 _amountRedeemed bool _userRedeemedEverything	onlyMensa
updateStateOnFlashLoan	address _reserve uint256 _availableLiquidityBefore uint256 _income uint256 _protocolFee	onlyMensa
updateStateOnBorrow	address _reserve address _user uint256 _amountBorrowed uint256 _borrowFee CoreLibrary.InterestRateMode _rateMode	onlyMensa
updateStateOnRepay	address _reserve address _user uint256 _paybackAmountMinusFees uint256 _originationFeeRepaid uint256 _balanceIncrease bool _repaidWholeLoan	onlyMensa

updateStateOnSwapRate	address _reserve address _user uint256 _principalBorrowBalance uint256 _compoundedBorrowBalance uint256 _balanceIncrease CoreLibrary.InterestRateMode _currentRateMode	onlyMensa
updateStateOnLiquidation	address _principalReserve address _collateralReserve address _user uint256 _amountToLiquidate uint256 _collateralToLiquidate uint256 _feeLiquidated uint256 _liquidatedCollateralForFee uint256 _balanceIncrease bool _liquidatorReceivesMToken	onlyMensaStaff
updateStateOnRebalance	address _reserve address _user uint256 _balanceIncrease	onlyMensa
setUserUseReserveAsCollateral	address _reserve address _user bool _useAsCollateral	onlyMensa
transferToUser	address _reserve address _user uint256 _amount	onlyMensaStaff
transferToFeeCollectionAddress	address _token address _user uint256 _amount address _destination	onlyMensa
liquidateFee	address _token uint256 _amount address _destination	onlyMensaStaff

transferToReserve	address _reserve address _user uint256 _amount	onlyMensaStaff
getUserBasicReserveData	address _reserve address _user	external
isUserAllowedToBorrowAtStable	address _reserve address _user uint256 _amount	external
getUserUnderlyingAssetBalance	address _reserve address _user	public
getReserveInterestRateStrategyAddress	address _reserve	public
getReserveUnit	address _reserve	public
getReserveMTokenAddress	address _reserve	public
getReserveAvailableLiquidity	address _reserve	public
getReserveTotalLiquidity	address _reserve	public
getReserveNormalizedIncome	address _reserve	external
getReserveTotalBorrows	address _reserve	public
getReserveTotalBorrowsStable	address _reserve	external
getReserveTotalBorrowsVariable	address _reserve	external
getReserveLiquidationThreshold	address _reserve	external
getReserveLiquidationBonus	address _reserve	external
getReserveCurrentVariableBorrowRate	address _reserve	external
getReserveCurrentStableBorrowRate	address _reserve	public
getReserveCurrentAverageStableBorrowRate	address _reserve	external
getReserveCurrentLiquidityRate	address _reserve	external
getReserveLiquidityCumulativeIndex	address _reserve	external
getReserveVariableBorrowsCumulativeIndex	address _reserve	external
getReserveConfiguration	address _reserve	external
getReserveDecimals	address _reserve	external
isReserveBorrowingEnabled	address _reserve	external
isReserveUsageAsCollateralEnabled	address _reserve	external
getReserveIsStableBorrowRateEnabled	address _reserve	external
getReserveIsActive	address _reserve	external

getReserveIsFreezed	address _reserve	external
getReserveLastUpdate	address _reserve	external
getReserveUtilizationRate	address _reserve	public
getReserves	none	external
isUserUseReserveAsCollateralEnabled	address _reserve address _user	external
getUserOriginationFee	address _reserve address _user	external
getUserCurrentBorrowRateMode	address _reserve address _user	public
getUserCurrentBorrowRate	address _reserve address _user	internal
getUserCurrentStableBorrowRate	address _reserve address _user	external
getUserBorrowBalances	address _reserve address _user	public
getUserVariableBorrowCumulativeIndex	address _reserve address _user	external
getUserLastUpdate	address _reserve address _user	external
refreshConfiguration	none	onlyMensaConfigurator
initReserve	address _reserve address _mTokenAddress uint256 _decimals address _interestRateStrategyAddress	onlyMensaConfigurator
removeLastAddedReserve	address _reserveToRemove	onlyMensaConfigurator
setReserveInterestRateStrategyAddress	address _reserve address _rateStrategyAddress	onlyMensaConfigurator
enableBorrowingOnReserve	address _reserve bool _stableBorrowRateEnabled	onlyMensaConfigurator
disableBorrowingOnReserve	address _reserve	onlyMensaConfigurator
enableReserveAsCollateral	address _reserve uint256	onlyMensaConfigurator

	_baseLTVasCollateral uint256 _liquidationThreshold uint256 _liquidationBonus	
disableReserveAsCollateral	address _reserve	onlyMensaConfigurator
enableReserveStableBorrowRate	address _reserve	onlyMensaConfigurator
disableReserveStableBorrowRate	address _reserve	onlyMensaConfigurator
activateReserve	address _reserve	onlyMensaConfigurator
deactivateReserve	address _reserve	onlyMensaConfigurator
freezeReserve	address _reserve	onlyMensaConfigurator
unfreezeReserve	address _reserve	onlyMensaConfigurator
setReserveBaseLTVasCollateral	address _reserve uint256 _ltv	onlyMensaConfigurator
setReserveLiquidationThreshold	address _reserve uint256 _threshold	onlyMensaConfigurator
setReserveLiquidationBonus	address _reserve uint256 _bonus	onlyMensaConfigurator
setReserveDecimals	address _reserve uint256 _decimals	onlyMensaConfigurator
updateReserveStateOnBorrowInternal	address _reserve address _user uint256 _principalBorrowBalance uint256 _balanceIncrease uint256 _amountBorrowed CoreLibrary.InterestRateMode _rateMode	internal
updateUserStateOnBorrowInternal	address _reserve address _user uint256 _amountBorrowed uint256 _balanceIncrease uint256 _fee	internal

	CoreLibrary.InterestRateMode _rateMode	
updateReserveStateOnRepayInternal	address _reserve address _user uint256 _paybackAmountMinusFees uint256 _balanceIncrease	internal
updateUserStateOnRepayInternal	address _reserve address _user uint256 _paybackAmountMinusFees uint256 _originationFeeRepaid uint256 _balanceIncrease bool _repaidWholeLoan	internal
updateReserveStateOnSwapRateInternal	address _reserve address _user uint256 _principalBorrowBalance uint256 _compoundedBorrowBalance CoreLibrary.InterestRateMode _currentRateMode	internal
updateUserStateOnSwapRateInternal	address _reserve address _user uint256 _balanceIncrease CoreLibrary.InterestRateMode _currentRateMode	internal
updatePrincipalReserveStateOnLiquidationInternal	address _principalReserve address _user uint256 _amountToLiquidate uint256 _balanceIncrease	internal
updateCollateralReserveStateOnLiquidationInternal	address _collateralReserve	internal

updateUserStateOnLiquidationInternal	address _reserve address _user uint256 _amountToLiquidate uint256 _feeLiquidated uint256 _balanceIncrease	internal
updateReserveStateOnRebalanceInternal	address _reserve address _user uint256 _balanceIncrease	internal
updateUserStateOnRebalanceInternal	address _reserve address _user uint256 _balanceIncrease	internal
updateReserveTotalBorrowsByRateModeInternal	address _reserve address _user uint256 _principalBalance uint256 _balanceIncrease uint256 _amountBorrowed CoreLibrary.InterestRateMode _newBorrowRateMode	internal
updateReserveInterestRatesAndTimestampInternal	address _reserve uint256 _liquidityAdded uint256 _liquidityTaken	internal
transferFlashLoanProtocolFeeInternal	address _token uint256 _amount	internal
refreshConfigInternal	none	internal
addReserveToListInternal	address _reserve	internal

DefaultReserveInterestRateStrategy contract

Name	Parameter	Attributes
getBaseVariableBorrowRate	none	external
getVariableRateSlope1	none	external

getVariableRateSlope2	none	external
getStableRateSlope1	none	external
getStableRateSlope2	none	external
calculateInterestRates	address _reserve uint256 _availableLiquidity uint256 _totalBorrowsStable uint256 _totalBorrowsVariable uint256 _averageStableBorrowRate	external
getOverallBorrowRateInternal	uint256 _totalBorrowsStable uint256 _totalBorrowsVariable uint256 _currentVariableBorrowRate uint256 _currentAverageStableBorrowRate	internal

IFlashLoanReceiver contract

Name	Parameter	Attributes
executeOperation	address _reserve uint256 _amount uint256 _fee bytes _params	external

FlashLoanReceiverBase contract

Name	Parameter	Attributes
transferFundsBackToPoolInternal	address _reserve uint256 _amount	internal
transferInternal	address _destination address _reserve uint256 _amount	internal
getBalanceInternal	address _target address _reserve	internal

4. Audit details

4.1 Risk distribution

Name	Risk level	Repair status
No events added	No	confirmed
Nested mapping problem	No	normal
Incorrect object initialization	No	normal
No error message added	No	confirmed
Arbitrary user bypass judgment	Low	confirmed
Variable update problem	No	normal
Integer Overflow	No	normal
Numerical accuracy	No	normal
Default visibility	No	normal
tx.origin authentication	No	normal
Wrong constructor	No	normal
Unverified return value	No	normal
Insecure random number	No	normal
Timestamp dependent	No	normal
Transaction order dependence	No	normal
Delegatecall	No	normal
Call	No	normal
Denial of service	No	normal
Logical design flaws	No	normal
Fake recharge vulnerability	No	normal
Short address attack	No	normal
Uninitialized storage pointer	No	normal
Frozen account bypass	No	normal
Uninitialized	No	normal
Reentry attack	No	normal

4.2 Risk audit details

4.2.1 No events added

- **Risk Description**

InterestRateOracle contract, getAddress, setMarketLiquidityRate method, MensaManager contract initAddressProvider, initReserve, initReserveWithData removeLastAddedReserve, enableBorrowingOnReserve , disableBorrowingOnReserve, enableReserveAsCollateral, disableReserveAsCollateral activateReserve methods, there are sensitive operations in several methods of the MensaMiner contract, but no events have been added. In order to facilitate administrators and users to understand the relevant operations, it is recommended to add method event logs, as shown in the following code:

```
    function disableBorrowingOnReserve(address _reserve) external onlyO
wner {
        conf.disableBorrowingOnReserve(_reserve);
    }

    function enableReserveAsCollateral(
        address _reserve,
        uint256 _baseLTVasCollateral,
        uint256 _liquidationThreshold,
        uint256 _liquidationBonus
    ) external onlyOwner {
        conf.enableReserveAsCollateral(_reserve,_baseLTVasCollateral,_l
iquidationThreshold,_liquidationBonus);
    }

    function disableReserveAsCollateral(address _reserve) external only
Owner {
        conf.disableReserveAsCollateral(_reserve);
    }

    function enableReserveStableBorrowRate(address _reserve) external o
nlyOwner {
        conf.enableReserveStableBorrowRate(_reserve);
    }

    function disableReserveStableBorrowRate(address _reserve) external
onlyOwner {
        conf.disableReserveStableBorrowRate(_reserve);
    }

    function activateReserve(address _reserve) external onlyOwner {
        conf.activateReserve(_reserve);
    }

    function deactivateReserve(address _reserve) external onlyOwner {
```

```
        conf.deactivateReserve(_reserve);
    }
    function createPool(uint256 _pid, uint256 _poolCap, uint256 _startB
lock, uint256 _endBlock) public noReentrancy onlyOwner {
        require(_pid == poolLength(), "createPool: _pid fault");
        require(block.number < _endBlock && _endBlock > _startBlock , "
createPool: Invailed block parameters");
        _poolCap = _poolCap.mul(1e18);
        require(_poolCap > 0, "createPool: cap fault");

        _alloced = _alloced.add(_poolCap);
        uint256 startBlock = block.number > _startBlock ? block.number
 : _startBlock;

        require(_endBlock > startBlock , "createPool: Invailed block pa
rameters 2");
        uint256 _bonusPerBlock = _poolCap.div(_endBlock.sub(startBloc
k));

        poolInfo.push(PoolInfo({
            poolCap: _poolCap,
            totalAllocPoint: 0,
            startBlock: startBlock,
            endBlock: _endBlock,
            bonusPerBlock: _bonusPerBlock,
            lockedTotal: 0,
            ageout: false
        }));
    }
    function deposit(uint256 _pid, uint256 _gid, address u, uint256 _am
ount) external noReentrancy onlyOwner {
        require(_pid < liquidityPool, "only for const pool");
        require(constPoolIsInit[_pid] == false, "init failed");
        constPoolIsInit[_pid] = true;
        _deposit(_pid, _gid, u, _amount);
    }
    function withdraw(uint256 _pid, uint256 _gid, address u, uint256 _a
mount) external noReentrancy onlyOwner {
        require(_pid < liquidityPool, "only for const pool" );
        _withdraw(_pid, _gid, u, _amount, true);
    }
```

- **Safety advice**

In order to facilitate administrators and users to understand the relevant operations, it is recommended to add method event logging.

- **Repair Status**

The risk has been fixed through communication with the Mensa project team.

4.2.2 Nested mapping problem

- **Risk description**

Nested mapping values are generally key-value pairs. Some contracts confuse the ke y input parameters when the nested mapping values are obtained, resulting in incor rect or abnormal values, causing the contract to fail to operate normally.

- **Audit result: passed**

4.2.3 Incorrect object initialization

- **Risk description**

The variable object in the contract has not changed its state, and using storage to initialize the variable will greatly increase gas consumption. This operation has the risk of exceeding the upper limit and rolling back, and will increase the risk of memory overwriting.

- **Audit result: passed**

4.2.4 No error message added

- **Risk description**

stake contract, take method, line 586 use require to determine whether the transfer of line 586 is successful, but there is no corresponding error prompt in require, as shown in the following code.

```
function stake(uint _MensaAmount) external  {
        _requireNonZeroAmount(_MensaAmount);
        uint currentStake = stakes[msg.sender];
        uint[64] memory gains;
        if (currentStake != 0) {
            for (uint i = 0; i < assetAddresses.length; i++) {
                gains[i] = _getPendingGain(msg.sender, assetAddresses
[i]);}
        }
        updateAssetsFeePerStake();
        _updateUserSnapshots();
        uint newStake = currentStake.add(_MensaAmount);
        stakes[msg.sender] = newStake;
        totalMensaStaked = totalMensaStaked.add(_MensaAmount);
        emit TotalMensaStakedUpdated(totalMensaStaked);
        bool succ = xensaToken.transferFrom(msg.sender, address(this),
_MensaAmount);
        require(succ);
        emit StakeChanged(msg.sender, newStake);
        emit Stake(msg.sender, _MensaAmount, block.timestamp);
        if (currentStake != 0) {
```

```
            if (gains[0] > 0) {
                _sendETHGainToUser(gains[0]);
                assetStakedTable[assetAddresses[0]].withdrawed = assetS
takedTable[assetAddresses[0]].withdrawed.add(gains[0]);}
            for (uint i = 1; i < assetAddresses.length; i++) {
                if (gains[i] > 0) {
                    IERC20 token = IERC20(assetAddresses[i]);
                    succ = token.transfer(msg.sender, gains[i]);
                    require(succ);
                    assetStakedTable[assetAddresses[i]].withdrawed = as
setStakedTable[assetAddresses[i]].withdrawed.add(gains[i]);}
            }
            emit assetSent(msg.sender, gains[0], gains[1], gains[2], ga
ins[3], gains[4]);}
    }
```

- **Safety advice**

It is recommended to add a transfer failure alert in require.

- **Repair Status**

The risk has been fixed through communication with the Mensa project team.

4.2.5 Arbitrary user bypass judgment

- **Risk description**

MensaMinter contract, withdrawPendingMensaToken method is called for any user, the two parameters passed are (uint256_gid, bool unlockedOnly), when the poolLength() > constPoolCount condition is constant, can further call withdraw method, as shown in the following code.

```
    function _withdrawPendingMensaToken(uint256 _gid, bool unlockedOnl
y) public {
        require(poolLength() > constPoolCount, "minMensaToken: Invailed
 pools");
        require(_gid == depositWithdraw || _gid == borrowRepay, "minMen
saToken: Invailed action");
        for (uint i = poolLength()-1; i >=  liquidityPool; i--) {
            if (poolInfo[i].groups[_gid].users[msg.sender].lastWithdraw
Block < block.number) {
                _withdraw(i, _gid, msg.sender, 0, unlockedOnly);
            }
        }
    }
    function _withdraw(uint256 _pid, uint256 _gid, address u, uint256 _
amount, bool unlockedOnly) internal {
        WithdrawVar memory vars;
        PoolInfo storage pool = poolInfo[_pid];
```

```
        if (pool.ageout == false && pool.endBlock < block.number) {
            pool.ageout = true;
        }
        Group storage group = pool.groups[_gid];
        UserInfo storage user = group.users[u];
        if (!(group.totalAmount > 0)) {
            return;
        }
        require(user.amount >= _amount, "withdraw: not good");
        if (pool.endBlock < block.number){
            user.protectAmount = 0;
            user.protectMintPending = 0;
        }
        vars.unlockAmount = user.amount.sub(user.protectAmount);
        require(!(unlockedOnly && vars.unlockAmount<_amount) , "withdra
w: unlocked balance not enough");
        vars.multiplier = getMultiplier(user.lastWithdrawBlock, block.n
umber, pool.startBlock, pool.endBlock);
        vars.pending = vars.multiplier.mul(group.accPerShare).mul(user.
amount).div(group.totalAmount);
        vars.protectPrice;
        {
            (vars.protectPrice, vars.blackHole) = userGainPrice(_pid, _
gid, u, _amount, false);
            if (_amount>0 && vars.unlockAmount >= _amount) {
                unlockedOnly = true;
            }
        }
        (vars.fine, vars.flush) = updateUserProtectDuration(u, user, 0,
 vars.pending, vars.protectPrice, unlockedOnly);
        pool.lockedTotal = pool.lockedTotal.add(vars.fine.div(3).mul
(2)).sub(vars.protectPrice).sub(vars.blackHole);
        user.amount = user.amount.sub(_amount);
        group.totalAmount = group.totalAmount.sub(_amount);
        if (vars.fine > 0) {
            updatePricePerStake(_pid, vars.fine.div(3).mul(2));
            mint(address(0xdead), vars.fine.div(3));
        }
        if (user.protectAmount > user.amount){
            user.protectAmount = user.amount;
        }
        user.rewardDebt = user.rewardDebt.add(vars.pending);
        if (!pool.ageout) {
            updateGroup(_pid, _gid);
        }
        emit MintWithdraw(msg.sender, _pid, _gid, _amount, group.totalA
mount, vars.flush, vars.fine);
    }
```

In the above _withdraw method, there are two require judgments.

First: when user.amount is not less than 0, the (user.amount >= amount) condition holds, that is, when amount is equal to 0, the condition is constant.

Second: when unlockedOnly is false, the (! (unlockedOnly && vars.unlockAmount<_amount) has the possibility to hold

When the above two conditions are established, you can continue to execute other code, when there is no user call, you can also complete the above two conditions through controlled passing of parameters.

- **Safety advice**

Recommended judgment_ Whether the caller user of the withdrawpendingxensatoken method exists.

- **Repair Status**

The risk has been fixed through communication with the Mensa project team.

4.2.6 Variable update problem

- **Vulnerability description**

The variable update problem generally occurs in the reward and transfer stage. If a user gets the reward he deserves, but after the reward is sent, the variable of the reward is not updated in time in the contract, which causes the reward amount to always exist. If the vulnerability is attacked by the attacker Malicious use may lead to abnormal capital loss and market stability.

- **Audit result: passed**

4.2.7 Integer Overflow

- **Vulnerability description**

Integer overflow is generally divided into overflow and underflow. There are three types of integer overflow in smart contracts: multiplication overflow, addition overflow, and subtraction overflow. In the Solidity language, the integer type step length supported by the variable is incremented by 8, and it supports from uint8 to uint256, and int8 to int256. The integer specifies a fixed-size data type and is unsigned. For example, a uint8 type can only be stored Numbers in the range 0 to 2^8-1, which is [0,255], a uint256 type can only store numbers in the range 0 to 2^256-1. This means that an integer variable can only be represented by a certain range of numbers, and cannot exceed the specified range. Exceeding the range of values expressed by the variable type will result in an integer overflow vulnerability.

- **Audit result: passed**

4.2.8 Numerical accuracy

- **Vulnerability description**

Solidity does not support floating-point type, nor does it fully support fixed-length floating-point type. The result of division operation will be rounded up. If there is a decimal, the part after the decimal point will be discarded, and only the integer part will be taken, for example, use 5 directly. Divide by 2, the result is 2. If the calculation result of the token is less than 1, for example, 4.9 tokens will be roughly equal to 4, which will cause a certain degree of loss of accuracy. Due to the economic properties of tokens, the loss of precision is equivalent to the loss of assets, so this will bring about the problem of accumulating in the frequently traded token.

- **Audit result: passed**

4.2.9 Default visibility

- **Vulnerability description**

In Solidity, the visibility of contract functions is public by default. Therefore, functions that do not specify any visibility can be called externally by the user. When a developer erroneously ignores the visibility specifier of a function that should be private, or a visibility specifier that can only be called within the contract itself, it will lead to serious vulnerabilities. In the first hack of the Parity multi-signature wallet, it was because the visibility of the function was not set, and the default was public, which led to the theft of a large amount of funds.

- **Audit result: passed**

4.2.10 tx.origin authentication

- **Vulnerability description**

tx.origin is a global variable of Solidity that traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in a smart contract makes the contract vulnerable to attacks like phishing.

- **Audit result: passed**

4.2.11 Wrong constructor

- **Vulnerability description**

Before the 0.4.22 version of the solidity smart contract, all contracts and constructors had the same name. When writing a contract, if the constructor function name is not the same as the contract name, the contract will add a default constructor function, and the constructor function set by yourself will be treated as a normal function, causing your original contract settings to not execute as expected, , especially if the constructor is performing a privileged operation.

- **Audit result: passed**

4.2.12 Unverified return value

- **Vulnerability description**

There are three methods to send tokens to an address in Solidity: transfer(), send(), call.value(). The difference between them is that when the transfer function fails to send, it throws an exception throw, rolls back the transaction status, and costs 2300gas; when the send function fails to send, it returns false and costs 2300gas; when the call.value method fails to send, it returns false, and the call costs all gas. Will lead to the risk of re-entry attacks. If the send or call.value method is used in the contract code to send the token without checking the method return value, if an error occurs, the contract will continue to execute the subsequent code, which will cause unexpected results.

- **Audit result: passed**

4.2.13 Insecure random number

- **Vulnerability description**

All transactions on the blockchain are deterministic state transition operations without uncertainty, which ultimately means that there is no source of entropy or randomness in the blockchain ecosystem. So there is no random number function like rand() in Solidity. Many developers use future block variables, such as block hash value, timestamp, block height, or gas upper limit to generate random numbers. These quantities are controlled by the miners, so they are not truly random. , So using past or current block variables to generate random numbers may lead to destructive vulnerabilities.

- **Audit result: passed**

4.2.14 Timestamp dependent

- **Vulnerability description**

In the blockchain, the data block timestamp (block.timestamp) is used in various applications, such as the function of random numbers, the locking of funds for a period of time, and the conditional statements of various state changes related to time. Miners have the ability to adjust the timestamp according to their needs. For example, block.timestamp or the alias now can be manipulated by the miners. If the wrong block timestamp is used in the smart contract, this can lead to serious vulnerabilities. If the contract is not particularly concerned about the miner's manipulation of the block timestamp, this may be unnecessary, but this should be paid attention to when developing the contract.

- **Audit result: passed**

4.2.15 Transaction order dependence

- **Vulnerability description**

In the blockchain, the miners choose the transaction with the highest transaction fee and pack it into the block. Since the transaction information in the block is public, the attacker can observe whether there are transactions in the transaction pool that may contain a solution to the problem, modify or revoke the attacker's authority or change the state of the contract that is unfavorable to the attacker. Then, the attacker can obtain data from this transaction and create a higher-level transaction gasPrice and include its transaction in a block before the original, which will preempt the original transaction solution.

- **Audit result: passed**

4.2.16 Delegatecall

- **Vulnerability description**

In Solidity, the delegatecall function is a standard message calling method, but the code in the target address will run in the environment of the calling contract, that is, keep msg.sender and msg.value unchanged. This function supports the implementation of the library, and developers can create reusable code for future contracts. The code in the library itself can be safe and flawless, but when running in another application environment, new vulnerabilities may occur, so using the delegatecall function may cause unexpected code execution.

- **Audit result: passed**

4.2.17 Call

- **Vulnerability description**

The call function is similar to the delegatecall function. They are both low-level functions provided by the smart contract writing language Solidity to interact with external contracts or libraries. However, when the call function method is used to process the call to the external standard information of the contract, the code is in the external contract/ Run in a functional environment. When using this type of function, it is necessary to determine the security of the calling parameters. It is recommended to use it with caution. Attackers can easily borrow the identity of the current contract to perform other malicious operations, leading to serious vulnerabilities.

- **Audit result: passed**

4.2.18 Denial of service

- **Vulnerability description**

There are a wide range of reasons for denial-of-service attacks, and its purpose is to allow users to make the contract unable to function normally for a period of time or permanently under certain circumstances, including malicious behavior when acting as a transaction receiver, and artificially increasing the gas required for computing functions. Leading to gas exhaustion, abusing access control to access the private components of the contract, the owner with privileges in the contract is modified, based on external calls, using obfuscation, etc. can lead to denial of service attacks.

- **Audit result: passed**

4.2.19 Logical design flaws

- **Vulnerability description**

In smart contracts, the special functions designed by developers for their own contracts are intended to stabilize the market value of tokens or the life of the project, and increase the highlights of the project. However, the more complex the system, the more likely it is to make mistakes. It is precisely in these logic and In the function, a slight error may lead to a serious deviation between the whole logic and the expectation, leaving fatal hidden dangers, such as logic judgment errors, function implementation and design inconsistency etc.

- **Audit result: passed**

4.2.20 Fake recharge vulnerability

- **Vulnerability description**

The success or failure of the token transaction status (true or false) depends on whether an exception is thrown during the execution of the transaction (for example, the require/assert/revert/throw mechanism is used). When the user calls the transfer function of the token contract to transfer, if the transfer function runs normally and no exception is thrown, regardless of whether the transfer transaction is successful or not, the receipt status of the transaction is successful or true. Then the transfer function of some token contracts uses the if judgment method to check the balance of the transfer initiator (msg.sender). When balances[msg.sender] <_value, it enters the else logic part and returns false, and finally no exception is thrown. , But the transaction receipt is successful, then we believe that only the mild judgment method of if/else is an imprecise coding method in sensitive function scenarios such as transfer, which will lead to related centralized exchanges, centralized wallets, and Fake recharge loopholes in token contracts.

- **Audit result: passed**

4.2.21 Short address attack

- **Vulnerability description**

In the solid smart contract, when parameters are passed to the smart contract, the parameters will be coded according to ABI specification. EVM running attackers send encoding parameters shorter than expected. For example, when transferring money in an exchange or a wallet, you need to send the transfer address and the transfer amount value. An attacker can send a 19 byte address instead of a standard 20 byte address. In this case, EVM will fill 0 in the end of the encoding parameter to make up the expected length, which will lead to the overflow of the final transfer amount parameter value and change the original transfer amount.

- **Audit result: passed**

4.2.22 Uninitialized storage pointer

- **Vulnerability description**

EVM uses both storage and memory to store variables. Local variables in functions are stored in storage or memory by default according to their types. In the working mode of solid, state variables are stored in slots of contracts according to the order in which they appear in contracts, Uninitialized local storage variables may point to other unexpected storage variables in the contract, resulting in intentional or unintentional vulnerabilities.

- **Audit result: passed**

4.2.23 Frozen account bypass

- **Vulnerability description**

In the transfer operation code of the contract, it detects whether the logic function of checking the freezing status of the transfer account exists in the contract code, and if the transfer account has been frozen, whether it can be bypassed.

- **Audit result: passed**

4.2.24 Uninitialized

- **Vulnerability description**

The initialize function in the contract can be called by other attackers before grabbing owner, so as to initialize governor.

- **Audit result: passed**

4.2.25 Reentry attack

- **Vulnerability description**

The attacker constructs a contract containing malicious code at the external address of fallback function. When the contract sends tokens to this address, it will call malicious code. When the call. Value() function in solid is used to send tokens, it will consume all the gas it receives, Therefore, when the call. Value() function is called to send tokens before the actual reduction of the sender's account balance, a reentry attack will occur. Due to the reentry vulnerability, the Dao attack is well-known.

- **Audit result: passed**

5. Security Audit Tool

Tool name	Tool Features
Oyente	Can be used to detect common bugs in smart contracts
securify	Common types of smart contracts that can be verified
MAIAN	Multiple smart contract vulnerabilities can be found and classified
Noneage Internal Toolkit	Noneage(hawkeye system) self-developed toolkit + https://audit.noneage.com

Disclaimer：

Noneage Technology only issues a report and assumes corresponding responsibilities for the facts that occurred or existed before the issuance of this report, Since the facts that occurred after the issuance of the report cannot determine the security status of the smart contract, it is not responsible for this.

Noneage Technology conducts security audits on the security audit items in the project agreement, and is not responsible for the project background and other circumstances, The subsequent on-chain deployment and operation methods of the project party are beyond the scope of this audit.

This report only conducts a security audit based on the information provided by the information provider to Noneage at the time the report is issued, If the information of this project is concealed or the situation reflected is inconsistent with the actual situation, Noneage Technology shall not be liable for any losses and adverse effects caused thereby.

There are risks in the market, and investment needs to be cautious. This report only conducts security audits and results announcements on smart contract codes, and does not make investment recommendations and basis.

Email : support@noneage.com

Site : www.noneage.com

Weibo : weibo.com/noneage