MahaDAO

ARTH Loans

Smart Contract Audit Report

August 10, 2021

IMMUNE BYTES

Introduction

1. About MahaDAO

ARTH is a new type of currency designed to not be pegged to government-owned currencies (like US Dollar, Euro, or Chinese Yuan), but still remain relatively stable (unlike Gold and Bitcoin).

Without being influenced by government-owned currencies, ARTH will be immune to inflation. Through stability, ARTH also becomes a superior choice of currency for means of trade. This is unlike Gold or Bitcoin, which are used more as a store of value rather than a medium of exchange.

Visit http://mahadao.com/ to learn more about.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provide professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, dydx.

The team has been able to secure 15+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

Visit http://immunebytes.com/ to know more about the services.

Documentation Details

The MahaDAO team has provided the following doc for the purpose of audit:

1. https://docs.liquity.org/

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

- Project Name: MahaDAO
- Contracts Name: TroveManager, StabilityPool, ActivePool, BorrowerOperations
- Languages: Solidity(Smart contract), Javascript(Unit Testing)
- Github commit hash for audit: 117c1005adb4ad8e443a4f3e803adb539a128cf2
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck, Fuzz

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Security Level References

Every issue in this report was assigned a severity level from the following:

Admin/Owner Privileges can be misused either intentionally or unintentionally.

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	High	Medium	Low
Open	1	6	4
Closed	-	-	-

Contract Name: TroveManager

High severity issues

1. **Trove's Status is not adequately validated in the _getTotalsFromBatchLiquidate_NormalMode function**
   Line no: 967-998

   Explanation
   The protocol uses a **batchLiquidateTroves** function that is used to liquidate a custom list of troves. This function includes two imperative internal functions, i.e., **_getTotalFromBatchLiquidate_RecoveryMode** & **_getTotalFromBatchLiquidate_NormalMode** which are used when the batch liquidation sequence starts during Recovery Mode or Normal mode respectively.

   As far as the **_getTotalFromBatchLiquidate_RecoveryMode** function is concerned, it does include adequate validation to ensure that the Troves passed as an argument are in an **Active status** (Line 908-910), before proceeding with the further execution in the function body. All the Non-Active troves are skipped.

```
904
905         for (vars.i = 0; vars.i < _troveArray.length; vars.i++) {
906             vars.user = _troveArray[vars.i];
907             // Skip non-active troves
908             if (Troves[vars.user].status != Status.active) {
909                 continue;
910             }
911             vars.ICR = getCurrentICR(vars.user, _price);
912
```

_getTotalFromBatchLiquidate_RecoveryMode function

   However, no such validation was found in the **_getTotalFromBatchLiquidate_NormalMode** function. This leads to an unexpected scenario where even the Non Active troves are forwarded for further execution in the function body.

```
978
979         for (vars.i = 0; vars.i < _troveArray.length; vars.i++) {
980             vars.user = _troveArray[vars.i];
981             vars.ICR = getCurrentICR(vars.user, _price);
982
```

_getTotalFromBatchLiquidate_NormalMode function

This issue is handled effectively when the **batchLiquidateTroves** function is called from the external function **liquidate()**. This is because the **liquidate** function uses a require statement at Line 329 to validate the argument passed and ensure that the Trove is Active for the passed address.

```
328     function liquidate(address _borrower) external override {
329         _requireTroveIsActive(_borrower);
330
331         address[] memory borrowers = new address[](1);
332         borrowers[0] = _borrower;
333         batchLiquidateTroves(borrowers);
334     }
335
```

However, since the **batchLiquidateTroves** is a **Public** function, it can be called individually without triggering the **liquidate** function as well and therefore should include all the imperative and relevant validations itself.

```
807
808     /*
809      * Attempt to liquidate a custom list of troves provided by the caller.
810      */
811     function batchLiquidateTroves(address[] memory _troveArray) public override {
812         require(_troveArray.length != 0, "TroveManager: Calldata address array must not be empty");
813
```

Recommendation

If the above-mentioned scenario is not intended or was not considered while designing the function, it is recommended to include adequate and necessary validations in the function for all the arguments passed to it before proceeding with further execution.

Medium severity issues

1. **Redundant Local variable used in Function. Adverse effect on Gas Optimization**
   Line no - 366

   Explanation
   Keeping in mind the bulky size of the TroveManager contract, the protocol uses **Variable container structs** which are used to assign hold or return variables in the liquidation functions of the contract.

   This is done to avoid any **Stack too Deep** scenarios as well as effectively manage the Gas optimization for the protocol.

   However, the _liquidateNormalMode at Line 366 unnecessarily uses a local variable **collToLiquidate** instead of using the already available vars.**collToLiquidate** from the **LocalVariables_InnerSingleLiquidateFunction** struct.

```
365         singleLiquidation.LUSDGasCompensation = LUSD_GAS_COMPENSATION;
366         uint256 collToLiquidate = singleLiquidation.entireTroveColl.sub(
367             singleLiquidation.collGasCompensation
368         );
369
370         (
```

_liquidateNormalMode function in TroveManager

   While this depicts a redundant use of the local variable **collToLiquidate**, it also adversely affects the gas optimization factor of the function.

   Recommendation
   Considering the fact that the TroveManager.sol contract is quite bulky in nature, every possible step must be taken to optimize the gas usage in the protocol.

   The above mentioned issue can be resolved by simply using the **collToLiquidate** with the help of already defined structs.

   For instance, the **_liquidateRecoveryMode** function (at Line 417) implements this same variable in a comparatively effective manner and thus can be taken as reference to modify the **_liquidateNormalMode** function.

```
416         singleLiquidation.LUSDGasCompensation = LUSD_GAS_COMPENSATION;
417         vars.collToLiquidate = singleLiquidation.entireTroveColl.sub(
418             singleLiquidation.collGasCompensation
419         );
420
```

_liquidateRecoveryMode function TroveManager

2. **Multiplication is being performed on the result of Division**
   Line no - 1534-1546

   Explanation
   During the automated testing of the **TroveManager** contract, it was found that some of the functions in the contract are performing multiplication on the result of a Division.
   Integer Divisions in Solidity might truncate. Moreover, this performing division before multiplication might lead to loss of precision.

   The following functions involve division before multiplication in the mentioned lines:
   - **_redistributeDebtAndColl** at 1534-1546

   Automated Test Results for the above-mentioned functions

```
TroveManager._redistributeDebtAndColl(IActivePool,IDefaultPool,uint256,uint256) (myFlats/FlatTrove.sol#2899-2944) performs a multiplication on the result of a division:
        -ETHRewardPerUnitStaked = ETHNumerator.div(totalStakes) (myFlats/FlatTrove.sol#2926)
        -lastETHError_Redistribution = ETHNumerator.sub(ETHRewardPerUnitStaked.mul(totalStakes)) (myFlats/FlatTrove.sol#2929)
TroveManager._redistributeDebtAndColl(IActivePool,IDefaultPool,uint256,uint256) (myFlats/FlatTrove.sol#2899-2944) performs a multiplication on the result of a division:
        -LUSDDebtRewardPerUnitStaked = LUSDDebtNumerator.div(totalStakes) (myFlats/FlatTrove.sol#2927)
        -lastLUSDDebtError_Redistribution = LUSDDebtNumerator.sub(LUSDDebtRewardPerUnitStaked.mul(totalStakes)) (myFlats/FlatTrove.sol#2930-2932)
```

   Recommendation
   Solidity doesn't encourage arithmetic operations that involve division before multiplication. Therefore the above-mentioned function should be checked once and redesigned if they do not lead to expected results.

Low severity issues

No issues found

Informational

1. Coding Style Issues in the TroveManager

Explanation

Code readability of a smart contract is largely influenced by the Coding Style issues and in some specific scenarios may lead to bugs in the future.

```
Parameter TroveManager.setTroveStatus(address,uint256)._borrower (myFlats/FlatTrove.sol#3266) is not in mixedCase
Parameter TroveManager.setTroveStatus(address,uint256)._num (myFlats/FlatTrove.sol#3266) is not in mixedCase
Parameter TroveManager.increaseTroveColl(address,uint256)._borrower (myFlats/FlatTrove.sol#3271) is not in mixedCase
Parameter TroveManager.increaseTroveColl(address,uint256)._collIncrease (myFlats/FlatTrove.sol#3271) is not in mixedCase
Parameter TroveManager.decreaseTroveColl(address,uint256)._borrower (myFlats/FlatTrove.sol#3282) is not in mixedCase
Parameter TroveManager.decreaseTroveColl(address,uint256)._collDecrease (myFlats/FlatTrove.sol#3282) is not in mixedCase
Parameter TroveManager.increaseTroveDebt(address,uint256)._borrower (myFlats/FlatTrove.sol#3293) is not in mixedCase
Parameter TroveManager.increaseTroveDebt(address,uint256)._debtIncrease (myFlats/FlatTrove.sol#3293) is not in mixedCase
Parameter TroveManager.decreaseTroveDebt(address,uint256)._borrower (myFlats/FlatTrove.sol#3304) is not in mixedCase
Parameter TroveManager.decreaseTroveDebt(address,uint256)._debtDecrease (myFlats/FlatTrove.sol#3304) is not in mixedCase
Variable TroveManager.Troves (myFlats/FlatTrove.sol#1464) is not in mixedCase
Variable TroveManager.L_ETH (myFlats/FlatTrove.sol#1482) is not in mixedCase
Variable TroveManager.L_LUSDDebt (myFlats/FlatTrove.sol#1483) is not in mixedCase
Variable TroveManager.TroveOwners (myFlats/FlatTrove.sol#1495) is not in mixedCase
Variable TroveManager.lastETHError_Redistribution (myFlats/FlatTrove.sol#1498) is not in mixedCase
Variable TroveManager.lastLUSDDebtError_Redistribution (myFlats/FlatTrove.sol#1499) is not in mixedCase
```

During the automated testing, it was found that the **TroveManager** contract had quite a few code style issues.

Recommendation

Therefore, it is recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.

Contract Name: StabilityPool

High severity issues

No issues found

Medium severity issues

1. **Contract includes functions that perform Multiplication on the result of Division.**

   Explanation
   As per the automated test results of **StabilityPool** contract, the functions **_computeLQTYPerUnitStaked** & **_computeRewardsPerUnitStaked** perform multiplication on the result of division.

   Integer Divisions in Solidity might truncate. Moreover, this performing division before multiplication might lead to loss of precision.

   The functions with the specific line numbers are mentioned below:
   - **_computeLQTYPerUnitStaked** at 525-526
   - **_computeRewardsPerUnitStaked** at 596-597

   Automated Test Results for the above-mentioned functions

```
StabilityPool._computeLQTYPerUnitStaked(uint256,uint256) (myFlats/FlatStability.sol#2021-2042) performs a multiplication on the result of a division:
        -LQTYPerUnitStaked = LQTYNumerator.div(_totalLUSDDeposits) (myFlats/FlatStability.sol#2038)
        -lastLQTYError = LQTYNumerator.sub(LQTYPerUnitStaked.mul(_totalLUSDDeposits)) (myFlats/FlatStability.sol#2039)
StabilityPool._computeRewardsPerUnitStaked(uint256,uint256,uint256) (myFlats/FlatStability.sol#2073-2113) performs a multiplication on the result of a division:
        -ETHGainPerUnitStaked = ETHNumerator.div(_totalLUSDDeposits) (myFlats/FlatStability.sol#2109)
        -lastETHError_Offset = ETHNumerator.sub(ETHGainPerUnitStaked.mul(_totalLUSDDeposits)) (myFlats/FlatStability.sol#2110)
```

   Recommendation
   Solidity doesn't encourage arithmetic operations that involve division before multiplication.
   Therefore the above-mentioned functions should be checked once and redesigned if they do not lead to expected results.

Low severity issues

1. **Contract's WETH balance is not checked before initiating a transfer**
   Line no - 899-909

   Explanation
   The **_sendEthGainToDepositor** function is responsible for transferring a particular amount of WETH to the caller.
   This internal function is used in some crucial functions like **provideToSP() or withdrawFromSP()**.

```
899     function _sendETHGainToDepositor(uint256 _amount) internal {
900         if (_amount == 0) {
901             return;
902         }
903         uint256 newETH = ETH.sub(_amount);
904         ETH = newETH;
905         emit StabilityPoolETHBalanceUpdated(newETH);
906         emit EtherSent(msg.sender, _amount);
907
908         weth.transfer(msg.sender, _amount);
909     }
```

   However, the **weth** transfer in this function is executed without considering whether or not the contract has an adequate amount of **WETH** in the first place.

   This validation might be quite imperative in scenarios when the WETH balance in the contract is not enough to execute this transfer.

   Recommendation
   It would be quite effective to include a validation that ensures that it has the adequate amount of WETH to complete a transfer.
   Moreover, including this validation will also help users to get a clear understanding behind a failed transfer in case of the above-mentioned scenario.

2. **Return Value of an External Call is not used Effectively**
   Line no - 908, 919, 1082

   Explanation
   The external calls made in the above-mentioned lines do return a boolean value that indicates whether or not the external call made was successful.

These boolean return values can be used in the function as a check to ensure that the further execution of the function is only allowed if the external is successfully made.

```
1079
1080        function receiveETH(uint256 _amount) external override {
1081            _requireCallerIsActivePool();
1082            weth.transferFrom(msg.sender, address(this), _amount);
1083            ETH = ETH.add(_amount);
1084            emit StabilityPoolETHBalanceUpdated(ETH);
1085        }
```

However, the **StabilityPool** contract never uses these return values to ensure the adequate execution of external calls.

Recommendation
Effective use of all the return values from external calls must be ensured within the contract.

Informational

1. **Coding Style Issues in StabilityPool**

Explanation
Code readability of a smart contract is largely influenced by the coding style issues and in some specific scenarios may lead to bugs in the future.

```
Parameter StabilityPool.provideToSP(uint256,address)._amount (myFlats/FlatStability.sol#1857) is not in mixedCase
Parameter StabilityPool.provideToSP(uint256,address)._frontEndTag (myFlats/FlatStability.sol#1857) is not in mixedCase
Parameter StabilityPool.withdrawFromSP(uint256)._amount (myFlats/FlatStability.sol#1906) is not in mixedCase
Parameter StabilityPool.withdrawETHGainToTrove(address,address)._upperHint (myFlats/FlatStability.sol#1952) is not in mixedCase
Parameter StabilityPool.withdrawETHGainToTrove(address,address)._lowerHint (myFlats/FlatStability.sol#1952) is not in mixedCase
Parameter StabilityPool.offset(uint256,uint256)._debtToOffset (myFlats/FlatStability.sol#2051) is not in mixedCase
Parameter StabilityPool.offset(uint256,uint256)._collToAdd (myFlats/FlatStability.sol#2051) is not in mixedCase
Parameter StabilityPool.getDepositorETHGain(address)._depositor (myFlats/FlatStability.sol#2196) is not in mixedCase
Parameter StabilityPool.getDepositorLQTYGain(address)._depositor (myFlats/FlatStability.sol#2242) is not in mixedCase
Parameter StabilityPool.getFrontEndLQTYGain(address)._frontEnd (myFlats/FlatStability.sol#2274) is not in mixedCase
Parameter StabilityPool.getCompoundedLUSDDeposit(address)._depositor (myFlats/FlatStability.sol#2324) is not in mixedCase
Parameter StabilityPool.getCompoundedFrontEndStake(address)._frontEnd (myFlats/FlatStability.sol#2343) is not in mixedCase
Parameter StabilityPool.registerFrontEnd(uint256)._kickbackRate (myFlats/FlatStability.sol#2439) is not in mixedCase
Parameter StabilityPool.receiveETH(uint256)._amount (myFlats/FlatStability.sol#2593) is not in mixedCase
Variable StabilityPool.ETH (myFlats/FlatStability.sol#1682) is not in mixedCase
Variable StabilityPool.P (myFlats/FlatStability.sol#1720) is not in mixedCase
Variable StabilityPool.lastETHError_Offset (myFlats/FlatStability.sol#1752) is not in mixedCase
```

During the automated testing, it was found that the contract had quite a few code style issues.

Recommendation:
It's recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.

Contract Name: ActivePool

High severity issues

No issues found

Medium severity issues

1. **Violation of Check_Effects_Interaction Pattern in the Withdraw function**
   Line no - 149-155

   Explanation
   The **fallback function**(receiveETH) in the **ActivePool** contract updates state variables after the external call is being made and therefore violates the Check Effects Interaction Pattern.

```
149 ▾    function receiveETH(uint256 _amount) external override {
150          _requireCallerIsBorrowerOperationsOrDefaultPool();
151          weth.transferFrom(msg.sender, address(this), _amount);
152          ETH = ETH.add(_amount);
153          emit ActivePoolETHBalanceUpdated(ETH);
154      }
```

   An external call within a function technically shifts the control flow of the contract to another contract for a particular period of time.
   Therefore, as per the Solidity Guidelines, any modification of the state variables in the base contract must be performed before executing the external call.

   Recommendation
   Check Effects Interaction Pattern must be followed while implementing external calls in a function.

2. **Unchecked WETH Transfers found in Contract**
   Line no: 97, 151

   Explanation
   The external calls made in the above-mentioned lines do return a boolean value that indicates whether or not the external call made was successful.

   These boolean return values can be used in the function as a check to ensure that the further execution of the function is only allowed if the external is successfully made.

However, the **ActivePool** contract never uses these return values throughout the contract, to validate if transfers were successful.

Recommendation
The return values should be used effectively in the function.

Low severity issues

No issues found

Informational

1. **Coding Style Issues in ActivePool Contract**

Explanation
Code readability of a smart contract is largely influenced by the Coding Style issues and in some specific scenarios may lead to bugs in the future.

```
Parameter ActivePool.setAddresses(address,address,address,address,address,address)._borrowerOperationsAddress (myFlats/FlatActive.sol#466) is not in mixedCase
Parameter ActivePool.setAddresses(address,address,address,address,address,address)._troveManagerAddress (myFlats/FlatActive.sol#467) is not in mixedCase
Parameter ActivePool.setAddresses(address,address,address,address,address,address)._stabilityPoolAddress (myFlats/FlatActive.sol#468) is not in mixedCase
Parameter ActivePool.setAddresses(address,address,address,address,address,address)._defaultPoolAddress (myFlats/FlatActive.sol#469) is not in mixedCase
Parameter ActivePool.setAddresses(address,address,address,address,address,address)._collSurplusPoolAddress (myFlats/FlatActive.sol#470) is not in mixedCase
Parameter ActivePool.setAddresses(address,address,address,address,address,address)._wethAddress (myFlats/FlatActive.sol#471) is not in mixedCase
Parameter ActivePool.sendETH(address,uint256)._account (myFlats/FlatActive.sol#512) is not in mixedCase
Parameter ActivePool.sendETH(address,uint256)._amount (myFlats/FlatActive.sol#512) is not in mixedCase
Parameter ActivePool.increaseLUSDDebt(uint256)._amount (myFlats/FlatActive.sol#526) is not in mixedCase
Parameter ActivePool.decreaseLUSDDebt(uint256)._amount (myFlats/FlatActive.sol#532) is not in mixedCase
Parameter ActivePool.receiveETH(uint256)._amount (myFlats/FlatActive.sol#571) is not in mixedCase
Variable ActivePool.ETH (myFlats/FlatActive.sol#453) is not in mixedCase
Variable ActivePool.LUSDDebt (myFlats/FlatActive.sol#454) is not in mixedCase
```

During the automated testing, it was found that the ActivePool contract had some code style issues.

Recommendation
It's recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.

Contract Name: BorrowerOperations

High severity issues

No issues found

Medium severity issues

1. **Redundant assert validation in setAddresses function**
   Line no: 130

   Explanation
   During the manual code review it was found that the **setAddresses** function in the BorrowerOperations contract includes an **assert** validation to ensure that the state variable **MIN_NET_DEBT** is greater than Zero whenever the function is called.

```
115     function setAddresses(
116         address _troveManagerAddress,
117         address _activePoolAddress,
118         address _defaultPoolAddress,
119         address _stabilityPoolAddress,
120         address _gasPoolAddress,
121         address _collSurplusPoolAddress,
122         address _sortedTrovesAddress,
123         address _lusdTokenAddress,
124         address _lqtyStakingAddress,
125         address _wethAddress,
126         address _governanceAddress,
127         address _coreControllerAddress
128     ) external override onlyOwner {
129         // This makes impossible to open a trove with zero withdrawn LUSD
130         assert(MIN_NET_DEBT > 0);
131
```

   However, no significance for this validation was found as the **MIN_NET_DEBT** is not an argument passed to the function but a state variable that is already initialized with a value greater than Zero in the **LiquityBase** contract.

```
32      // Minimum amount of net LUSD debt a trove must have
33      uint256 public constant MIN_NET_DEBT = 1800e18;
34      // uint constant public MIN_NET_DEBT = 0;
35
```

MIN_NET_DEBT state variable in LiquityBase Contract

Recommendation
If the above-mentioned function design is not an intended one, it should be modified so that redundant validations are removed and gas usage is optimized.

Low severity issues

1. **_activePoolAddColl function ignores the Return value from external call**
   Line no: 584

   Explanation
   The activePoolAddColl function doesn't take into consideration the return value from **weth** transfers.

```
583     function _activePoolAddColl(IActivePool _activePool, uint256 _amount) internal {
584         weth.transferFrom(msg.sender, address(this), _amount);
585         weth.approve(address(_activePool), _amount);
586         _activePool.receiveETH(_amount);
587     }
588
```

   These return values could be effectively used to ensure that the external calls made within the function body were successful.

   Recommendation
   Return values shouldn't be ignored and must be used effectively.

2. **BorrowerOperations contract includes unused Internal Functions**
   Line no: 521-525, 627-632

   Explanation
   During the manual code review of the BorrowerOperations contract, it was found that it includes some internal functions that are never used throughout the contract.

   Moreover, since these functions have been assigned an **internal** visibility, they cannot be accessed from outside the contracts and can only be called from within the contract.

Now, since these functions are not being called from within the contract, they depict no significant use case and unnecessarily consume the contract's space.

Following functions are not being used in the contract:

1. _requireCallerIsBorrower

```
627     function _requireCallerIsBorrower(address _borrower) internal view {
628         require(
629             msg.sender == _borrower,
630             "BorrowerOps: Caller must be the borrower for a withdrawal"
631         );
632     }
```

2. _getUSDValue

```
521     function _getUSDValue(uint256 _coll, uint256 _price) internal pure returns (uint256) {
522         uint256 usdValue = _price.mul(_coll).div(DECIMAL_PRECISION);
523
524         return usdValue;
525     }
```

Recommendation

Functions with no significant use should be removed from the contract.

Informational

1. **Coding Style Issues in BorrowerOperations contract**

Explanation

Code readability of a smart contract is largely influenced by the Coding Style issues and in some specific scenarios may lead to bugs in the future.

```
Parameter BorrowerOperations.openTrove(uint256,uint256,uint256,address,address)._lowerHint (myFlats/FlatBorrower.sol#1722) is not in mixedCase
Parameter BorrowerOperations.addColl(uint256,address,address)._ETHAmount (myFlats/FlatBorrower.sol#1810) is not in mixedCase
Parameter BorrowerOperations.addColl(uint256,address,address)._upperHint (myFlats/FlatBorrower.sol#1811) is not in mixedCase
Parameter BorrowerOperations.addColl(uint256,address,address)._lowerHint (myFlats/FlatBorrower.sol#1812) is not in mixedCase
Parameter BorrowerOperations.moveETHGainToTrove(uint256,address,address,address)._ETHAmount (myFlats/FlatBorrower.sol#1819) is not in mixedCase
Parameter BorrowerOperations.moveETHGainToTrove(uint256,address,address,address)._borrower (myFlats/FlatBorrower.sol#1820) is not in mixedCase
Parameter BorrowerOperations.moveETHGainToTrove(uint256,address,address,address)._upperHint (myFlats/FlatBorrower.sol#1821) is not in mixedCase
Parameter BorrowerOperations.moveETHGainToTrove(uint256,address,address,address)._lowerHint (myFlats/FlatBorrower.sol#1822) is not in mixedCase
Parameter BorrowerOperations.withdrawColl(uint256,address,address)._collWithdrawal (myFlats/FlatBorrower.sol#1830) is not in mixedCase
Parameter BorrowerOperations.withdrawColl(uint256,address,address)._upperHint (myFlats/FlatBorrower.sol#1831) is not in mixedCase
Parameter BorrowerOperations.withdrawColl(uint256,address,address)._lowerHint (myFlats/FlatBorrower.sol#1832) is not in mixedCase
Parameter BorrowerOperations.withdrawLUSD(uint256,uint256,address,address)._maxFeePercentage (myFlats/FlatBorrower.sol#1839) is not in mixedCase
Parameter BorrowerOperations.withdrawLUSD(uint256,uint256,address,address)._LUSDAmount (myFlats/FlatBorrower.sol#1840) is not in mixedCase
Parameter BorrowerOperations.withdrawLUSD(uint256,uint256,address,address)._upperHint (myFlats/FlatBorrower.sol#1841) is not in mixedCase
Parameter BorrowerOperations.withdrawLUSD(uint256,uint256,address,address)._lowerHint (myFlats/FlatBorrower.sol#1842) is not in mixedCase
Parameter BorrowerOperations.repayLUSD(uint256,address,address)._LUSDAmount (myFlats/FlatBorrower.sol#1849) is not in mixedCase
Parameter BorrowerOperations.repayLUSD(uint256,address,address)._upperHint (myFlats/FlatBorrower.sol#1850) is not in mixedCase
Parameter BorrowerOperations.repayLUSD(uint256,address,address)._lowerHint (myFlats/FlatBorrower.sol#1851) is not in mixedCase
```

During the automated testing, it was found that the contract had quite a few code style issues.

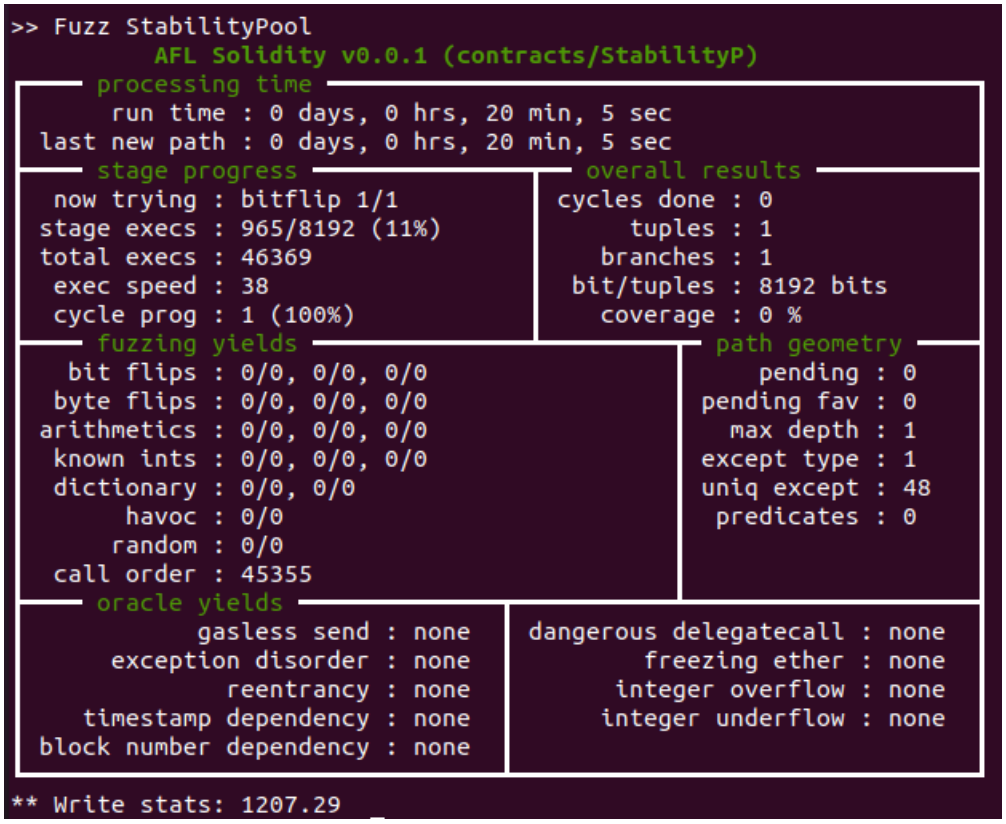
Recommendation

Therefore, it is recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.

Fuzz Testing

1. StabilityPool.sol: -

    a. Terminal Output
    [With use of : " -g -r 0 -d 1200 "]

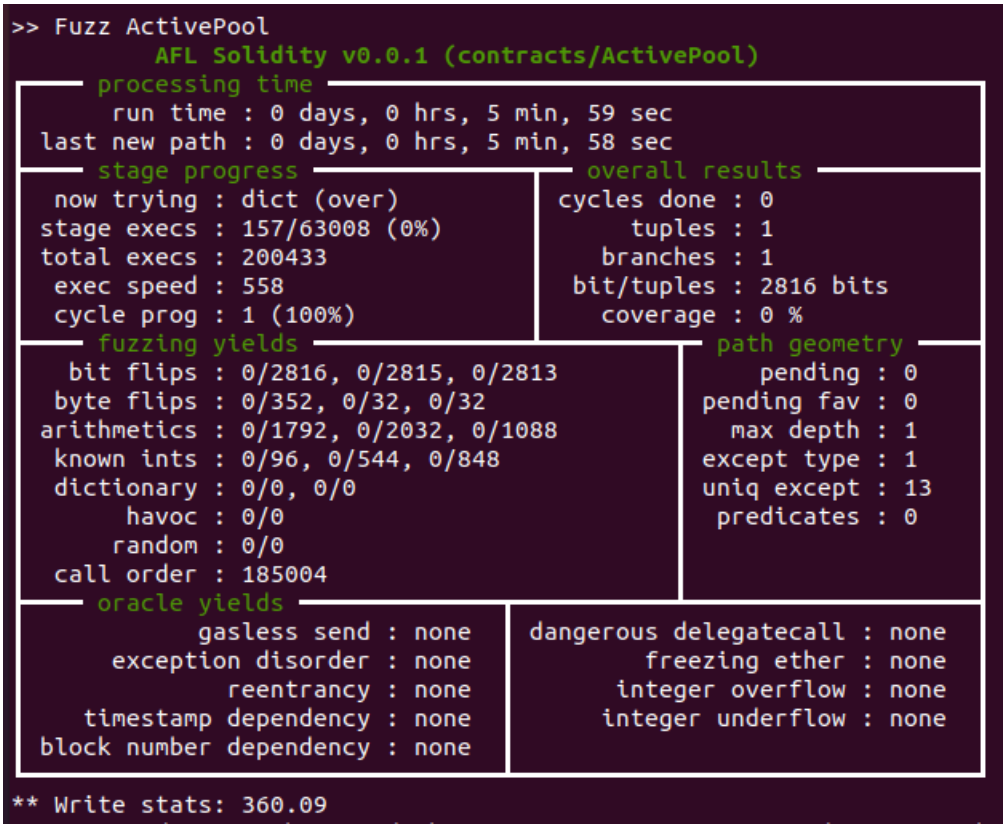


- Excel Sheet of States for the Output of Fuzz Testing
  [With use of : " -g -r 1 -d 1200 "]

  https://drive.google.com/file/d/1It-HVbQh4D2rsNRvEfPmUTi-QRwQd6QI/view?usp=sharing

2. ActivePool.sol: -

    a. Terminal Output
    [With use of : " -g -r 0 -d 360 "]

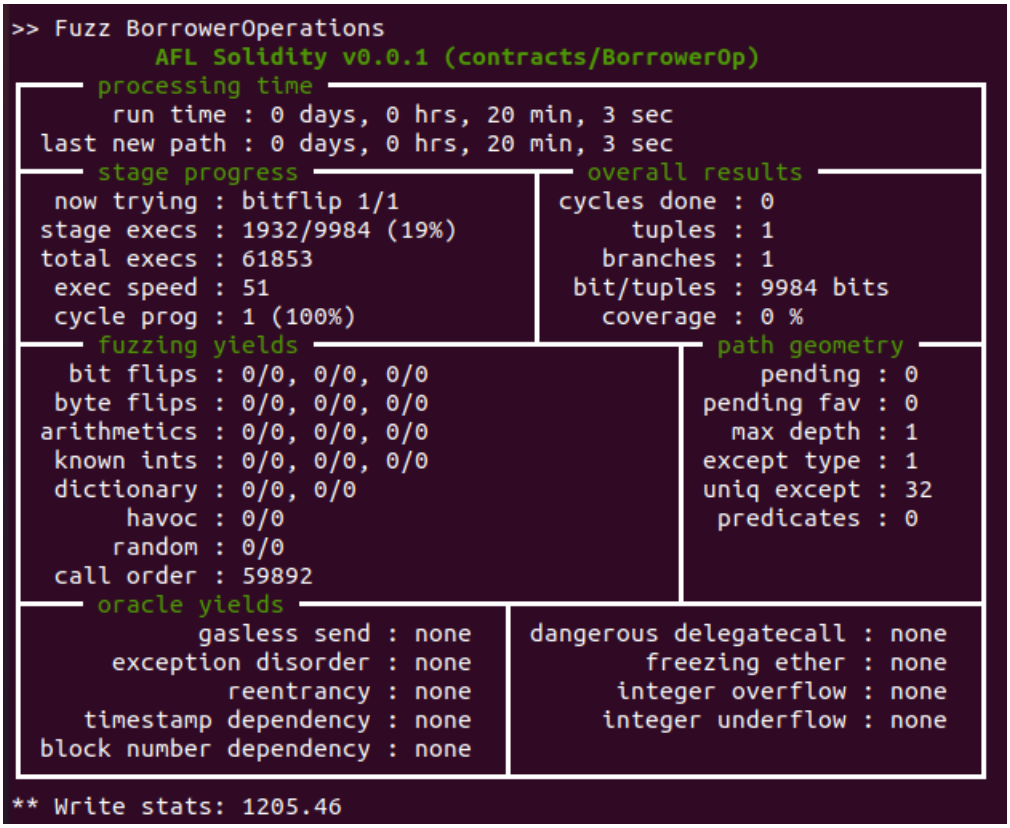


- Excel Sheet of States for the Output of Fuzz Testing
  [With use of : " -g -r 1 -d 360 "]

  https://drive.google.com/file/d/1oZ9wBxhxgp5OUanZr9jXTDvKV93z26WX/view?usp=sharing

3. BorrowerOpersations.sol: -

    a. Terminal Output
    [With use of : " -g -r 0 -d 1200 "]



- Excel Sheet of States for the Output of Fuzz Testing
  [With use of : " -g -r 1 -d 1200 "]

  https://drive.google.com/file/d/1cbWQeMviNA1XBzmhG1WELTakzT_Kxt_Y/view?usp=sharing

Automated Audit Result

1. TroveManager

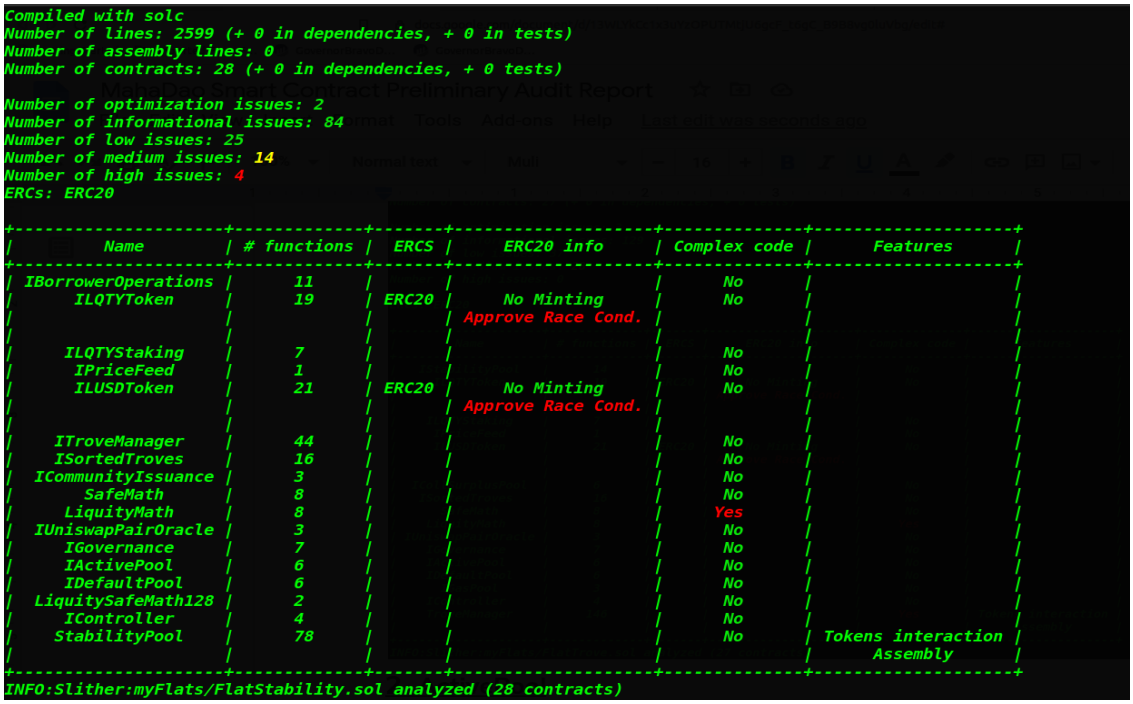
```
Compiled with solc
Number of lines: 3314 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 27 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 3
Number of informational issues: 129
Number of low issues: 37
Number of medium issues: 28
Number of high issues: 0

ERCs: ERC20

+--------------------+-------------+-------+--------------------+--------------+--------------------+
|        Name        | # functions | ERCS  |     ERC20 info     | Complex code |      Features      |
+--------------------+-------------+-------+--------------------+--------------+--------------------+
|   IStabilityPool   |      14     |       |                    |      No      |                    |
|     ILQTYToken     |      19     | ERC20 |     No Minting     |      No      |                    |
|                    |             |       | Approve Race Cond. |              |                    |
|                    |             |       |                    |              |                    |
|    ILQTYStaking    |      7      |       |                    |      No      |                    |
|     IPriceFeed     |      1      |       |                    |      No      |                    |
|     ILUSDToken     |      21     | ERC20 |     No Minting     |      No      |                    |
|                    |             |       | Approve Race Cond. |              |                    |
|                    |             |       |                    |              |                    |
|  ICollSurplusPool  |      6      |       |                    |      No      |                    |
|   ISortedTroves    |      16     |       |                    |      No      |                    |
|      SafeMath      |      8      |       |                    |      No      |                    |
|    LiquityMath     |      8      |       |                    |     Yes      |                    |
| IUniswapPairOracle |      3      |       |                    |      No      |                    |
|    IGovernance     |      7      |       |                    |      No      |                    |
|    IActivePool     |      6      |       |                    |      No      |                    |
|    IDefaultPool    |      6      |       |                    |      No      |                    |
|      IGasPool      |      3      |       |                    |      No      |                    |
|    IController     |      4      |       |                    |      No      |                    |
|    TroveManager    |     146     |       |                    |     Yes      | Tokens interaction |
|                    |             |       |                    |              |      Assembly      |
+--------------------+-------------+-------+--------------------+--------------+--------------------+
INFO:Slither:myFlats/FlatTrove.sol analyzed (27 contracts)
```

2. StabilityPool

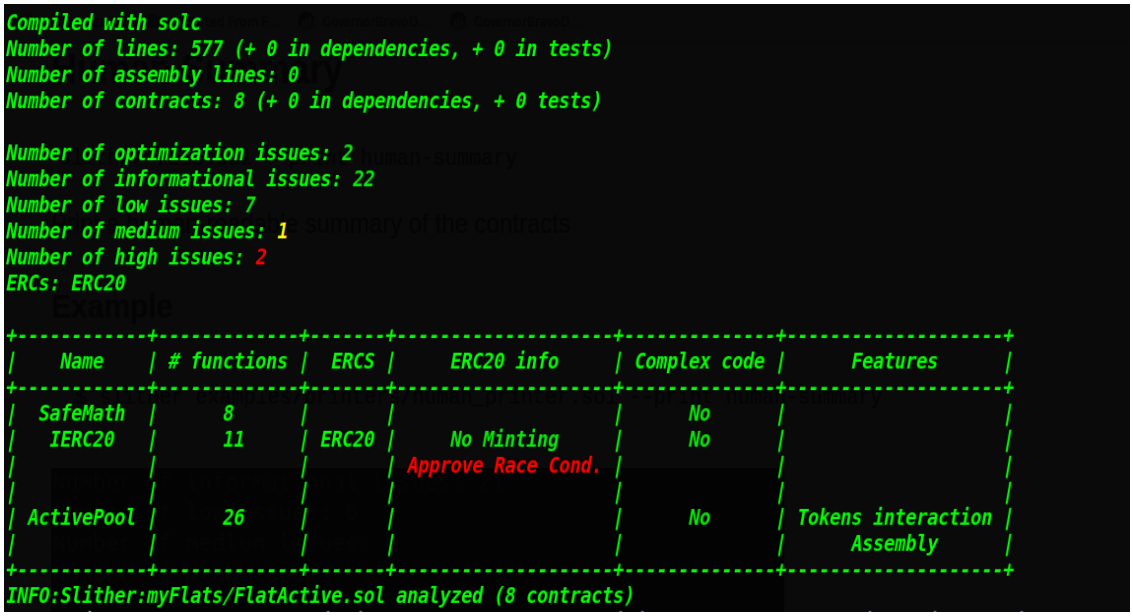


```
Compiled with solc
Number of lines: 2599 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 28 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 2
Number of informational issues: 84
Number of low issues: 25
Number of medium issues: 14
Number of high issues: 4
ERCs: ERC20

+----------------------+-------------+-------+--------------------+--------------+--------------------+
|         Name         | # functions | ERCS  |     ERC20 info     | Complex code |      Features      |
+----------------------+-------------+-------+--------------------+--------------+--------------------+
| IBorrowerOperations  |      11     |       |                    |      No      |                    |
|      ILQTYToken      |      19     | ERC20 |     No Minting     |      No      |                    |
|                      |             |       | Approve Race Cond. |              |                    |
|                      |             |       |                    |              |                    |
|     ILQTYStaking     |      7      |       |                    |      No      |                    |
|      IPriceFeed      |      1      |       |                    |      No      |                    |
|      ILUSDToken      |      21     | ERC20 |     No Minting     |      No      |                    |
|                      |             |       | Approve Race Cond. |              |                    |
|                      |             |       |                    |              |                    |
|    ITroveManager     |      44     |       |                    |      No      |                    |
|    ISortedTroves     |      16     |       |                    |      No      |                    |
|  ICommunityIssuance  |      3      |       |                    |      No      |                    |
|       SafeMath       |      8      |       |                    |      No      |                    |
|     LiquityMath      |      8      |       |                    |     Yes      |                    |
|  IUniswapPairOracle  |      3      |       |                    |      No      |                    |
|     IGovernance      |      7      |       |                    |      No      |                    |
|     IActivePool      |      6      |       |                    |      No      |                    |
|     IDefaultPool     |      6      |       |                    |      No      |                    |
|  LiquitySafeMath128  |      2      |       |                    |      No      |                    |
|     IController      |      4      |       |                    |      No      |                    |
|    StabilityPool     |      78     |       |                    |      No      | Tokens interaction |
|                      |             |       |                    |              |      Assembly      |
+----------------------+-------------+-------+--------------------+--------------+--------------------+
INFO:Slither:myFlats/FlatStability.sol analyzed (28 contracts)
```

3. ActivePool



```
Compiled with solc
Number of lines: 577 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 8 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 2
Number of informational issues: 22
Number of low issues: 7
Number of medium issues: 1
Number of high issues: 2
ERCs: ERC20

+------------+-------------+-------+--------------------+--------------+--------------------+
|    Name    | # functions | ERCS  |     ERC20 info     | Complex code |      Features      |
+------------+-------------+-------+--------------------+--------------+--------------------+
|  SafeMath  |      8      |       |                    |      No      |                    |
|   IERC20   |      11     | ERC20 |     No Minting     |      No      |                    |
|            |             |       | Approve Race Cond. |              |                    |
|            |             |       |                    |              |                    |
| ActivePool |      26     |       |                    |      No      | Tokens interaction |
|            |             |       |                    |              |      Assembly      |
+------------+-------------+-------+--------------------+--------------+--------------------+
INFO:Slither:myFlats/FlatActive.sol analyzed (8 contracts)
```

4. BorrowerOperations

```
Compiled with solc
Number of lines: 2408 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 28 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 2
Number of informational issues: 93
Number of low issues: 6
Number of medium issues: 7
Number of high issues: 1
ERCs: ERC20

+--------------------+-------------+-------+--------------------+--------------+--------------------+
|        Name        | # functions | ERCS  |     ERC20 info     | Complex code |      Features      |
+--------------------+-------------+-------+--------------------+--------------+--------------------+
|   IStabilityPool   |      14     |       |                    |      No      |                    |
|     ILQTYToken     |      19     | ERC20 |     No Minting     |      No      |                    |
|                    |             |       | Approve Race Cond. |              |                    |
|                    |             |       |                    |              |                    |
|    ILQTYStaking    |      7      |       |                    |      No      |                    |
|     IPriceFeed     |      1      |       |                    |      No      |                    |
|     ILUSDToken     |      21     | ERC20 |     No Minting     |      No      |                    |
|                    |             |       | Approve Race Cond. |              |                    |
|                    |             |       |                    |              |                    |
|   ITroveManager    |      44     |       |                    |      No      |                    |
|  ICollSurplusPool  |      6      |       |                    |      No      |                    |
|   ISortedTroves    |      16     |       |                    |      No      |                    |
|      SafeMath      |      8      |       |                    |      No      |                    |
|    LiquityMath     |      8      |       |                    |     Yes      |                    |
| IUniswapPairOracle |      3      |       |                    |      No      |                    |
|    IGovernance     |      7      |       |                    |      No      |                    |
|    IActivePool     |      6      |       |                    |      No      |                    |
|    IDefaultPool    |      6      |       |                    |      No      |                    |
|    IController     |      4      |       |                    |      No      |                    |
|      IGasPool      |      3      |       |                    |      No      |                    |
| BorrowerOperations |      74     |       |                    |      No      | Tokens interaction |
|                    |             |       |                    |              |      Assembly      |
+--------------------+-------------+-------+--------------------+--------------+--------------------+
```

Concluding Remarks

While conducting the audits of MahaDAO smart contracts(ARTH Loans), it was observed that the contracts contain High, Medium and Low severity issues along with a few areas of recommendations.

Our auditors suggest that High, Medium and Low severity issues should be resolved by MahaDAO developers. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the MahaDAO platform or its product nor this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes