AstraDAO

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **August 16th, 2021 - September 12th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	08/16/2021	Roberto Reigada
0.9	Document Updates	09/06/2021	Roberto Reigada
1.0	Final Review	09/13/2021	Gabi Urrutia
1.1	Remediation Plan	09/30/2021	Roberto Reigada

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Roberto Reigada	Halborn	Roberto.Reigada@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

AstraDAO engaged Halborn to conduct a security audit on their smart contracts beginning on August 16th, 2021 and ending on September 12th, 2021. The security assessment was scoped to the smart contracts provided in the Github repository AstraDAO repository

1.2 AUDIT SUMMARY

The team at Halborn was provided three weeks for the engagement and assigned a full time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks that were mostly addressed by the AstraDAO team.

HAL02 - SLASHING FEES/REDEPOSITS INCORRECT BEHAVIOUR was not solved yet as the fix would add too much complexity into the smart contracts. This issue only happens when a user redeposits and, for that reason, AstraDAO Team will educate their users and mention this edge case in their whitepaper to mitigate the risk. AstraDAO Team will consider implementing a fix in the Phase 2. The worst case scenario for this vulnerability is that a user does not follow AstraDAO's team advice, performs a re-deposit and then, when calling withdrawASTRReward() he receives less ASTR tokens than the amount he actually deserved.

On the other hand, Halborn wants to highlight the risks coming from HAL12 - WITHDRAW COOLDOWN PERIOD CAN BE BYPASSED. The potential issue here is caused if a user can deposit/withdraw in the same transaction as this could be abused with flash loans. With the current smart contracts code, even if a user bypassed the cooldown period and performed a flash loan, the user would not be able to benefit from it as the voting power would only increase 24 hours after the deposit. The 24 hour period should not be removed in the future otherwise this attack vector would be possible.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process,and implementation; automated testing techniques help enhance coverage of the bridge code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Brownie, Remix IDE)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident, and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

IN-SCOPE:

The security assessment was scoped to the smart contracts:

- poolv1.sol
- poolConfiguration.sol
- governance.sol
- oracle.sol
- itoken.sol
- timelock.sol
- chef.sol
- lm-pool.sol
- astr.sol

FIXED COMMIT ID: fbe94f26f6d3971b12b24b38cf2adaee19dfbef9

EXECUTIVE OVERVIEW

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	2	7	3

LIKELIHOOD

IMPACT	🟨 (HAL-03)	🟨	🟥 (HAL-01)	🟥	🟪
	🟩 (HAL-04)	🟨	🟨	🟥	🟥
	🟩 (HAL-05)	🟩	🟨 (HAL-02)	🟨	🟥
	⬜ (HAL-11)	🟩 (HAL-06) (HAL-08) (HAL-09) (HAL-10)	🟩 (HAL-07)	🟨	🟨
	⬜ (HAL-12) (HAL-13)	⬜	🟩	🟩	🟨

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - INTEGER OVERFLOW	High	SOLVED - 09/30/2021
HAL02 - SLASHING FEES/REDEPOSITS INCORRECT BEHAVIOUR	Medium	RISK ACCEPTED
HAL03 - FRONT-RUNNING ATTACK ON INITIALIZATION FUNCTIONS	Medium	SOLVED - 09/30/2021
HAL04 - UNCHECKED TRANSFER	Low	SOLVED - 09/30/2021
HAL05 - FLOATING PRAGMA	Low	RISK ACCEPTED
HAL06 - EXTERNAL CALLS WITHIN A LOOP	Low	SOLVED - 09/30/2021
HAL07 - MISSING ZERO ADDRESS CHECK	Low	SOLVED - 09/30/2021
HAL08 - VIOLATION OF CHECK, EFFECTS, INTERACTIONS PATTERN	Low	SOLVED - 09/30/2021
HAL09 - DIVIDE BEFORE MULTIPLY	Low	RISK ACCEPTED
HAL10 - TAUTOLOGY EXPRESSIONS	Low	SOLVED - 09/30/2021
HAL11 - USE OF INLINE ASSEMBLY	Informational	ACKNOWLEDGED
HAL12 - WITHDRAW COOLDOWN PERIOD CAN BE BYPASSED	Informational	ACKNOWLEDGED
HAL13 - TYPO IN FUNCTION AND VARIABLE	Informational	SOLVED - 09/30/2021

FINDINGS & TECH DETAILS

3.1 (HAL-01) INTEGER OVERFLOW - HIGH

Description:

In computer programming, an integer overflow occurs when an arithmetic operation attempts to create a numeric value that is outside of the range that can be represented with a given number of bits, either larger than the maximum or lower than the minimum value. Some of the operations in the contracts are using SafeMath correctly, other operations are not using SafeMath but make use of some of the SafeMath functions and others do not use any kind of SafeMath making the operations vulnerable to overflows and underflows.

Example - Proof of Concept:

UINT256_MAX_VALUE = 2ˆ256 - 1 = 115792089237316195423570985008687907853269984665640564039457584007913129639935

Listing 1: Overflow PoC in poolv1.sol contract (Lines 8)

```
PoolV1[0].addPublicPool([TESTERC20[5].address, TESTERC20[6].address], [1, UINT256_MAX_VALUE - 1], 0, 1000, "PublicPool", "PP1", "Public pool test", {'from': accounts[3]})
PoolV1[0].addPublicPool([TESTERC20[5].address, TESTERC20[6].address], [1, UINT256_MAX_VALUE], 0, 1000, "PublicPool2", "PP2", "Public pool test2", {'from': accounts[3]})

PoolV1[0].poolInfo(0)[0]
UINT256_MAX_VALUE

PoolV1[0].poolInfo(1)[0]
0
```

With this overflow an attacker could create a pool, force newTotalWeight to be 1 and then call the swap2() function retrieving more tokens than what he actually deserves:

Listing 2: poolv1.sol (Lines 872,883)

```
859 function swap2(address _token, uint _value, address[] memory
        newTokens, uint[] memory newWeights,uint newTotalWeight, uint[]
         memory _buf) internal returns(address[] memory, uint[] memory)
         {
860     // Use to get the share of particular token based on there
           share.
861     uint _tokenPart;
862     // Used to get the Expected amount for the token you are
           selling.
863     uint _amount;
864     buf = _buf;
865     // Used to get the distributing dex details for the token you
           are selling.
866     uint[] memory _distribution;
867     // Approve before selling the tokens
868     IERC20(_token).approve(EXCHANGE_CONTRACT, _value);
869      // Run loops over the tokens in the parametess to buy them.
870     for(uint i = 0; i < newTokens.length; i++) {
871
872         _tokenPart = _value.mul(newWeights[i]).div(newTotalWeight)
               ;
873
874         if(_tokenPart == 0) {
875             buf.push(0);
876             continue;
877         }
878
879         (_amount, _distribution) = IOneSplit(EXCHANGE_CONTRACT).
               getExpectedReturn(IERC20(_token), IERC20(newTokens[i]),
                _tokenPart, 2, 0);
880         uint256 minReturn = calculateMinimumRetrun(_amount);
881         buf.push(_amount);
882         newWeights[i] = _amount;
883         _amount= IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(_token),
                IERC20(newTokens[i]), _tokenPart, minReturn,
               _distribution, 0);
884     }
885     return (newTokens, newWeights);
886 }
```

Example - Vulnerable code:

poolv1.sol

Listing 3: poolv1.sol (Lines 181)

```
177 function calculateTotalWeight(uint[] memory _weights) internal view returns(uint){
178     uint _totalWeight;
179     // Calculate total weight for new index.
180     for(uint i = 0; i < _weights.length; i++) {
181         _totalWeight += _weights[i];
182     }
183     return _totalWeight;
184 }
```

Code Location:

We have located overflows in multiple contracts:

Listing 4: Overflows

```
version-5/governance.sol:297: if(block.timestamp<(startTime+7776000)){
version-5/governance.sol:326: proposalCount++;
version-5/governance.sol:528: uint256 oneday = proposalCreatedTime[proposalId]+6500;
version-5/governance.sol:588: votersInfo[proposalId].voterCount++;
version-5/governance.sol:590: votersInfo[proposalId].governors++;

version-5/poolv1.sol:181: _totalWeight += _weights[i];
version-5/poolv1.sol:243: _totalWeight += _weights[i];
version-5/poolv1.sol:343: tokenBalances[_poolIndex][returnedTokens[i]] += returnedAmounts[i];
version-5/poolv1.sol:650: _totalAmount += withdrawBalance;
version-5/poolv1.sol:660: _totalAmount += _amount;
version-5/poolv1.sol:714: _newTotalWeight += _weights[i];
version-5/poolv1.sol:813: _totalAmount += _amount;
version-5/poolv1.sol:907: _totalAmount += _amounts[i];
version-5/poolv1.sol:921: _totalAmount += _amount;

version-5/oracle.sol:231: TotalTokens = tokenLength+1;
```

```
version-5/oracle.sol:233: loopLenght = tokenLength*2+5;
version-5/oracle.sol:244: }else if(i==(loopLenght-2)){
version-5/oracle.sol:247: else if(i==(loopLenght-1)){
version-5/oracle.sol:325: uint256 _poolIndex = poolInfo.length - 1;

version-6/lm-pool.sol:384: eligibleAmount = eligibleAmount + stkInfo.amount;
version-6/lm-pool.sol:422: eligibleAmount = eligibleAmount + stkInfo.amount;
version-6/lm-pool.sol:869: uint256 day = (block.timestamp - currentUser.claimedTimestamp).div(dayseconds);
version-6/lm-pool.sol:891: uint256 day = (block.timestamp - currentUser.claimedTimestamp).div(dayseconds);

version-6/chef.sol:651: eligibleAmount = eligibleAmount + stkInfo.amount;
version-6/chef.sol:689: eligibleAmount = eligibleAmount + stkInfo.amount;
version-6/chef.sol:1483: uint256 day = (block.timestamp - currentUser.claimedTimestamp).div(dayseconds);
version-6/chef.sol:1505: uint256 day = (block.timestamp - currentUser.claimedTimestamp).div(dayseconds);
```

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

Currently not all the smart contracts and the operations within them are using the SafeMath library which makes some operations vulnerable to overflows/underflows. It is recommended to use the SafeMath library for arithmetic operations consistently throughout **ALL** the mathematical operations in the smart contract system.

Reference:

Ethereum Smart Contract Best Practices - Integer Overflow and Underflow

Remediation plan:

SOLVED: AstraDAO team now uses the SafeMath library to perform the mathematical operations.

Listing 5: poolv1.sol (Lines 213)

```
209 function calculateTotalWeight(uint[] memory _weights) internal
        view returns(uint){
210     uint _totalWeight;
211     // Calculate total weight for new index.
212     for(uint i = 0; i < _weights.length; i++) {
213         _totalWeight = _totalWeight.add(_weights[i]);
214     }
215     return _totalWeight;
216 }
```

Listing 6: Previous overflows now corrected

```
version-5/governance.sol:312: if(block.timestamp<add256(startTime
    ,7776000)){
version-5/governance.sol:342: proposalCount = add256(proposalCount
    ,1);
version-5/governance.sol:555: uint256 oneday = add256(
    proposalCreatedTime[proposalId],6500);
version-5/governance.sol:615: votersInfo[proposalId].voterCount =
    add256(votersInfo[proposalId].voterCount,1);
version-5/governance.sol:617: votersInfo[proposalId].governors =
    add256(votersInfo[proposalId].governors,1);

version-5/poolv1.sol:213: _totalWeight = _totalWeight.add(_weights
    [i]);
version-5/poolv1.sol:343: tokenBalances[_poolIndex][returnedTokens
    [i]] = tokenBalances[_poolIndex][returnedTokens[i]].add(
    returnedAmounts[i]);
version-5/poolv1.sol:650: _totalAmount = _totalAmount.add(
    withdrawBalance);
```

```
version-5/poolv1.sol:660: _totalAmount = _totalAmount.add(_amount);
version-5/poolv1.sol:715: _newTotalWeight = _newTotalWeight.add(_weights[i]);
version-5/poolv1.sol:813: _totalAmount = _totalAmount.add(_amount);
version-5/poolv1.sol:907: _totalAmount = _totalAmount.add(_amounts[i]);
version-5/poolv1.sol:921: _totalAmount = _totalAmount.add(_amount);

version-5/oracle.sol:233: TotalTokens = tokenLength.add(1);
version-5/oracle.sol:235: loopLenght = tokenLength.mul(2).add(5);
version-5/oracle.sol:246: }else if(i==(loopLenght.sub(2))){
version-5/oracle.sol:249: else if(i==(loopLenght.sub(1))){
version-5/oracle.sol:330: uint256 _poolIndex = poolInfo.length.sub(1);

version-6/lm-pool.sol:391: eligibleAmount = eligibleAmount.add(stkInfo.amount);
version-6/lm-pool.sol:429: eligibleAmount = eligibleAmount.add(stkInfo.amount);
version-6/lm-pool.sol:801: (block.timestamp.sub(currentUser.timestamp)).div(dayInSecond);
version-6/lm-pool.sol:873: uint256 day = block.timestamp.sub(currentUser.claimedTimestamp).div(dayseconds);

version-6/chef.sol:660: eligibleAmount = eligibleAmount.add(stkInfo.amount);
version-6/chef.sol:698: eligibleAmount = eligibleAmount = eligibleAmount.add(stkInfo.amount);
version-6/chef.sol:1379: (block.timestamp.sub(currentUser.timestamp)).div(dayInSecond);
version-6/chef.sol:1489: uint256 day = block.timestamp.sub(currentUser.claimedTimestamp).div(dayseconds);
```

3.2 (HAL-02) SLASHING FEES/REDEPOSITS INCORRECT BEHAVIOUR - MEDIUM

Description:

In the contracts chef.sol and lm-pool.sol there is a function called slashExitFee(). This function is called internally by the withdrawASTRReward() function. If the msg.sender has performed a deposit in the last 90 days slashExitFee() will reduce the percentage of ASTR tokens received by applying a slashing fee.

In regard to this function, Halborn has detected the following edge case:

ASTR total reward: 1000000 ASTR tokens distributed as INDIVIDUAL Reward.

Test 1

1. DAY 1: User1 deposits 200000e18 tokens in the 12 months vault
2. DAY 1: User2 deposits 200000e18 tokens in the 12 months vault
3. DAY 180: User1 redeposits another 200000e18 tokens in the 6 months vault
4. DAY 240: User1 calls withdrawASTRReward() and gets 466620 ASTR tokens
5. DAY 240: User2 calls withdrawASTRReward() and gets 399953 ASTR tokens

Test 2

1. DAY 1: User1 deposits 200000e18 tokens in the 12 months vault
2. DAY 1: User2 deposits 200000e18 tokens in the 12 months vault
3. DAY 240: User1 calls withdrawASTRReward() and gets 499900 ASTR tokens
4. DAY 240: User2 calls withdrawASTRReward() and gets 499900 ASTR tokens

This happened because the slashing fees were incorrectly applied to User1 total reward, instead of just the reward amount that belonged to the 200000e18 tokens that were deposited in the last 90 days prior to the call of withdrawASTRReward() function.

Code Location:

Listing 7: chef.sol (Lines 1452,1455,1460,1462)

```
1447 function slashExitFee(
1448     UserInfo storage currentUser,
1449     uint256 _pid,
1450     uint256 dayCount
1451 ) private {
1452     uint256 totalReward = currentUser.totalReward;
1453     uint256 sfr = uint256(90).sub(dayCount);
1454     // Here fee is calculated on the basis of how days is left in 90 days.
1455     uint256 fee = totalReward.mul(sfr).div(100);
1456     if (fee < 0) {
1457         fee = 0;
1458     }
1459     // Claimable reward is calculated by substracting the fee from total reward.
1460     uint256 claimableReward = totalReward.sub(fee);
1461     if (claimableReward > 0) {
1462         safeASTRTransfer(msg.sender, claimableReward);
1463         currentUser.totalReward = 0;
1464     }
1465     // Deducted fee would be distribute as reward to the same pool user as individual reward
1466     // with reward multiplier logic.
1467     distributeIndividualReward(_pid, fee);
1468     updateClaimedReward(currentUser, claimableReward);
1469 }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Redesign the slashExitFee() function so it takes into account this edge case, and in case of multiple deposits, the slashing fees are only applied to the rewards related to the deposits in the last 90 days.

Remediation Plan:

RISK ACCEPTED: AstraDAO Team accepts this risk as the fix would add too much complexity into the smart contracts. AstraDAO Team will educate their users and mention this edge case in their whitepaper so every one is aware of this issue. AstraDAO Team will consider implementing a fix in the Phase 2.

3.3 (HAL-03) FRONT-RUNNING ATTACK ON INITIALIZATION FUNCTIONS - MEDIUM

Description:

The contracts lm-pool.sol, chef.sol, poolConfiguration.sol, governance.sol, astr.sol, oracle.sol and poolv1.sol have initialization functions that can be front-run, allowing an attacker to incorrectly initialize the contracts.

For example, in the case of astr.sol, an attacker could front-run the initialize() call with a malicious transaction in which _allocationContract points to a contract owned by him:

Listing 8: astr.sol (Lines 15)

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.6.12;

import "./common/Address.sol";
import "./common/SafeMath.sol";
import "./common/Initializable.sol";
import "./upgrade/ERC20BurnableUpgradeSafe.sol";
import "./interface/ITransferHandler.sol";

contract Token is ERC20BurnableUpgradeSafe {

    address public allocationContract;

    function initialize(address _allocationContract) external
        initializer {
        Ownable.init(_allocationContract);
        __ERC20_init("Astra", "ASTRA");

        allocationContract = _allocationContract;

        _mint(allocationContract, 1000000000 * uint256(10)**
            decimals());
    }
}
```

Risk Level:

Likelihood - 1
Impact - 5

Recommendation:

Use a factory pattern that will deploy and initialize the contracts atomically to prevent front-running of the initialization.

Remediation Plan:

SOLVED: AstraDAO Team will make use of a factory pattern for the contracts deployment and initialization.

3.4 (HAL-04) UNCHECKED TRANSFER - LOW

Description:

In the contracts poolv1.sol, lm-pool.sol and chef.sol the return value of some external transfer/transferFrom calls are not checked. Several tokens do not revert in case of failure and return false. If one of these tokens is used, a deposit would not revert if the transfer fails, and an attacker could deposit tokens for free.

Code Location:

poolv1.sol

Listing 9: poolv1.sol

```
IERC20(baseStableCoin).transferFrom(msg.sender,address(this),
       stableValue);
```

Listing 10: poolv1.sol

```
IERC20(_tokens[0]).transferFrom(msg.sender,address(this),_values
       [0]);
```

Listing 11: poolv1.sol

```
transferTokens(baseStableCoin,msg.sender,totalAmount);
```

Listing 12: poolv1.sol

```
transferTokens(baseStableCoin,msg.sender,_pendingAmount);
```

Listing 13: poolv1.sol

```
transferTokens(baseStableCoin,managerAddresses,distribution);
```

Listing 14: poolv1.sol

```
606 transferTokens(baseStableCoin,poolInfo[_poolIndex].owner,
        distribution);
```

Listing 15: poolv1.sol

```
610 transferTokens(ASTRTokenAddress,address(poolChef),returnAmount);
```

Listing 16: poolv1.sol (Lines 617)

```
616 function transferTokens(address _token, address _reciever,uint
        _amount) internal{
617     IERC20(_token).transfer(_reciever, _amount);
618 }
```

lm-pool.sol

Listing 17: lm-pool.sol (Lines 476)

```
470 function safeASTRTransfer(address _to, uint256 _amount) internal {
471     uint256 ASTRBal = IERC20(ASTR).balanceOf(address(this));
472     require(
473         !(_amount > ASTRBal),
474         "Insufficient amount on lm pool contract"
475     );
476     IERC20(ASTR).transfer(_to, _amount);
477 }
```

chef.sol

Listing 18: chef.sol (Lines 740)

```
737 function safeASTRTransfer(address _to, uint256 _amount) internal {
738     uint256 ASTRBal = IERC20(ASTR).balanceOf(address(this));
739     require(!(_amount > ASTRBal), "Insufficient amount on chef
            contract");
740     IERC20(ASTR).transfer(_to, _amount);
741 }
```

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

It is recommended to use SafeERC20, or ensure that the transfer/transferFrom return value is checked.

Remediation Plan:

SOLVED: AstraDAO Team uses now the library SafeERC20.

3.5 (HAL-05) FLOATING PRAGMA - LOW

Description:

Contracts should be deployed with the same compiler version and flags used during development and testing. Locking the pragma helps to ensure that contracts do not accidentally get deployed using another pragma. For example, an outdated pragma version might introduce bugs that affect the contract system negatively or recently released pragma versions may have unknown security vulnerabilities.

Code Location:

Listing 19

```
# grep -Rin ”pragma solidity ^”
version-5/poolv1.sol:1:pragma solidity ^0.5.0;
version-5/poolConfiguration.sol:5:pragma solidity ^0.5.0;
version-5/oracle.sol:1:pragma solidity ^0.5.0;
version-5/itoken.sol:1:pragma solidity ^0.5.0;
version-5/timelock.sol:7:pragma solidity ^0.5.8;
version-6/chef.sol:4:pragma solidity ^0.6.6;
version-6/lm-pool.sol:1:pragma solidity ^0.6.6;
version-6/astr.sol:3:pragma solidity ^0.6.12;
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Consider locking the pragma version. It is not recommended to use a floating pragma in production. Apart from just locking the pragma version in the code, the sign (>=) need to be removed. It is possible to lock the pragma by fixing the version both in truffle-config.js for Truffle framework or in hardhat.config.js for HardHat framework.

Remediation Plan:

RISK ACCEPTED: AstraDAO Team accepts this risk.

3.6 (HAL-06) EXTERNAL CALLS WITHIN A LOOP - LOW

Description:

Calls inside a loop might lead to a Denial of Service attack. If the i variable iterates up to a very high value or is reset by the external functions called, this could cause a Denial of Service.

Code Location:

poolv1.sol

Listing 20: poolv1.sol (Lines 808,809)

```
799 function getPoolValue(uint256 _poolIndex)public view returns(uint256){
800     // Used to get the Expected amount for the token you are selling.
801     uint _amount;
802     // Used to get the distributing dex details for the token you are selling.
803     uint[] memory _distribution;
804     // Return the total Amount of Stable you will recieve for selling. This will be total value of pool that it has purchased.
805     uint _totalAmount;
806 
807     // Run loops over the tokens in the pool to get the token worth.
808     for (uint i = 0; i < poolInfo[_poolIndex].tokens.length; i++){
809         (_amount, _distribution) = IOneSplit(EXCHANGE_CONTRACT).getExpectedReturn(IERC20(poolInfo[_poolIndex].tokens[i]), IERC20(baseStableCoin), tokenBalances[_poolIndex][poolInfo[_poolIndex].tokens[i]], 2, 0);
810         if (_amount == 0) {
811             continue;
812         }
813         _totalAmount += _amount;
814     }
815 
816     // Return the total values of pool locked
817     return _totalAmount;
818 }
```

Listing 21: poolv1.sol (Lines 832,837,839,844,847)

```
824 function swap(address _token, uint _value, address[] memory _tokens, uint[] memory _weights, uint _totalWeight) internal returns(address[] memory, uint[] memory) {
825     // Use to get the share of particular token based on there share.
826     uint _tokenPart;
827     // Used to get the Expected amount for the token you are selling.
828     uint _amount;
829     // Used to get the distributing dex details for the token you are selling.
830     uint[] memory _distribution;
831        // Run loops over the tokens in the parametess to buy them.
832     for(uint i = 0; i < _tokens.length; i++) {
833         // Calculate the share of token based on the weight and the buy for that.
834         _tokenPart = _value.mul(_weights[i]).div(_totalWeight);
835 
836         // Get the amount of tokens pool will recieve based on the token selled.
837         (_amount, _distribution) = IOneSplit(EXCHANGE_CONTRACT).getExpectedReturn(IERC20(_token), IERC20(_tokens[i]), _tokenPart, 2, 0);
838         // calculate slippage
839         uint256 minReturn = calculateMinimumRetrun(_amount);
840         _weights[i] = _amount;
841 
842         // Check condition if token you are selling is ETH or another ERC20 and then sell the tokens.
843         if (_token == ETH_ADDRESS) {
844             _amount = IOneSplit(EXCHANGE_CONTRACT).swap.value(_tokenPart)(IERC20(_token), IERC20(_tokens[i]), _tokenPart, minReturn, _distribution, 0);
845         } else {
846             IERC20(_tokens[i]).approve(EXCHANGE_CONTRACT, _tokenPart);
847             _amount = IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(_token), IERC20(_tokens[i]), _tokenPart, minReturn, _distribution, 0);
848         }
849
850     }
851
852     return (_tokens, _weights);
853 }
```

Listing 22: poolv1.sol (Lines 870,879,880,883)

```
859 function swap2(address _token, uint _value, address[] memory newTokens, uint[] memory newWeights,uint newTotalWeight, uint[] memory _buf) internal returns(address[] memory, uint[] memory) {
860     // Use to get the share of particular token based on there share.
861     uint _tokenPart;
862     // Used to get the Expected amount for the token you are selling.
863     uint _amount;
864     buf = _buf;
865     // Used to get the distributing dex details for the token you are selling.
866     uint[] memory _distribution;
867     // Approve before selling the tokens
868     IERC20(_token).approve(EXCHANGE_CONTRACT, _value);
869      // Run loops over the tokens in the parametess to buy them.
870     for(uint i = 0; i < newTokens.length; i++) {
871
872         _tokenPart = _value.mul(newWeights[i]).div(newTotalWeight);
873
874         if(_tokenPart == 0) {
875             buf.push(0);
876             continue;
877         }
878
879         (_amount, _distribution) = IOneSplit(EXCHANGE_CONTRACT).getExpectedReturn(IERC20(_token), IERC20(newTokens[i]), _tokenPart, 2, 0);
880         uint256 minReturn = calculateMinimumRetrun(_amount);
881         buf.push(_amount);
882         newWeights[i] = _amount;
883         _amount= IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(_token), IERC20(newTokens[i]), _tokenPart, minReturn, _distribution, 0);
884     }
885     return (newTokens, newWeights);
886 }
```

Listing 23: poolv1.sol (Lines 901,914,920,923)

```
891 function sellTokensForStable(address[] memory _tokens, uint[] memory _amounts) internal returns(uint) {
892     // Used to get the Expected amount for the token you are selling.
893     uint _amount;
894     // Used to get the distributing dex details for the token you are selling.
895     uint[] memory _distribution;
896 
897     // Return the total Amount of Stable you will recieve for selling
898     uint _totalAmount;
899 
900     // Run loops over the tokens in the parametess to sell them.
901     for(uint i = 0; i < _tokens.length; i++) {
902         if (_amounts[i] == 0) {
903             continue;
904         }
905 
906         if (_tokens[i] == baseStableCoin) {
907             _totalAmount += _amounts[i];
908             continue;
909         }
910 
911         // Approve token access to Exchange contract.
912         IERC20(_tokens[i]).approve(EXCHANGE_CONTRACT, _amounts[i]);
913         // Get the amount of Stable tokens you will recieve for selling tokens
914         (_amount, _distribution) = IOneSplit(EXCHANGE_CONTRACT).getExpectedReturn(IERC20(_tokens[i]), IERC20(baseStableCoin), _amounts[i], 2, 0);
915         // Skip remaining execution if no token is available
916         if (_amount == 0) {
```

```
                continue;
            }
            // Calculate slippage over the the expected amount
            uint256 minReturn = calculateMinimumRetrun(_amount);
            _totalAmount += _amount;
            // Actually swap tokens
            _amount = IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(_tokens[i]), IERC20(baseStableCoin), _amounts[i], minReturn, _distribution, 0);


        }

        return _totalAmount;
}
```

lm-pool.sol

Listing 24: lm-pool.sol (Lines 615,617,621,623)

```
612 function distributeTvlAdjustedReward(uint256 _amount) private {
613     uint256 totalTvl = 0;
614     // Applied the loop for calculating the TVL(total value locked) and updating that in totalTvl variable.
615     for (uint256 pid = 0; pid < poolInfo.length; ++pid) {
616         PoolInfo storage pool = poolInfo[pid];
617         uint256 tvl = pool.lpToken.balanceOf(address(this));
618         totalTvl = totalTvl.add(tvl);
619     }
620     // Applied the loop for calculating the reward share for each pool and the distribute the share with all users.
621     for (uint256 pid = 0; pid < poolInfo.length; ++pid) {
622         PoolInfo storage pool = poolInfo[pid];
623         uint256 tvl = pool.lpToken.balanceOf(address(this));
624         uint256 poolRewardShare = tvl.mul(10000).div(totalTvl);
625         uint256 reward = (_amount.mul(poolRewardShare)).div(10000);
626         // After getting the pool reward share then it will same as individual reward.
627         distributeIndividualReward(pid, reward);
628     }
629 }
```

chef.sol

Listing 25: chef.sol (Lines 995,997,1001,1003)

```
992 function distributeTvlAdjustedReward(uint256 _amount) private {
993     uint256 totalTvl = 0;
994     // Applied the loop for calculating the TVL(total value locked) and updating that in totalTvl variable.
995     for (uint256 pid = 0; pid < poolInfo.length; ++pid) {
996         PoolInfo storage pool = poolInfo[pid];
997         uint256 tvl = pool.lpToken.balanceOf(address(this));
998         totalTvl = totalTvl.add(tvl);
999     }
1000    // Applied the loop for calculating the reward share for each pool and the distribute the share with all users.
1001    for (uint256 pid = 0; pid < poolInfo.length; ++pid) {
1002        PoolInfo storage pool = poolInfo[pid];
1003        uint256 tvl = pool.lpToken.balanceOf(address(this));
1004        uint256 poolRewardShare = tvl.mul(10000).div(totalTvl);
1005        uint256 reward = (_amount.mul(poolRewardShare)).div(10000);
1006        // After getting the pool reward share then it will same as individual reward.
1007        distributeIndividualReward(pid, reward);
1008    }
1009 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

If possible, use pull over push strategy for external calls.

Remediation Plan:

SOLVED: AstraDAO team added the following require statement in the addPublicPool() function:

Listing 26: poolv1.sol (Lines 245)

```
require (_tokens.length <= IPoolConfiguration(_poolConf).
    getmaxTokenSupported(), "E16");
```

IPoolConfiguration(_poolConf).getmaxTokenSupported() is set to return 10 in the poolConfiguration contract:

Listing 27: PoolConfiguration.sol (Lines 34)

```
// Maximum number of tokens supported by indices
uint256 private maxTokenSupported = 10;
```

Thanks to this require statement no pool can contain more than 10 tokens. This means that _tokens.length is limited to 10, so in the loops the external calls are actually limited now.

3.7 (HAL-07) MISSING ZERO ADDRESS CHECK - LOW

Description:

There is no validation of the addresses anywhere in the code. Every address should be validated and checked that is different than zero. This issue is present in all the smart contracts, in the constructors and functions that use addresses as parameters.

Some code location examples:

poolv1.sol

Listing 28: poolv1.sol (Lines 147,148)

```
145 constructor(address _ASTRTokenAddress, address poolConfiguration,
      address _itokendeployer, address _chef) public {
146     systemAddresses[msg.sender] = true;
147     ASTRTokenAddress = _ASTRTokenAddress;
148     managerAddresses = msg.sender;
149     _poolConf = poolConfiguration;
150     itokendeployer = _itokendeployer;
151     poolChef = _chef;
152 }
```

chef.sol

Listing 29: chef.sol (Lines 230)

```
229 function setLmPoolAddress(address _lmpooladdr) external onlyOwner
      {
230     lmpooladdr = _lmpooladdr;
231 }
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

Validate that every address input is different than zero.

Remediation Plan:

SOLVED: AstraDAO team added validation to every address input.

3.8 (HAL-08) VIOLATION OF CHECK, EFFECTS, INTERACTIONS PATTERN - LOW

Description:

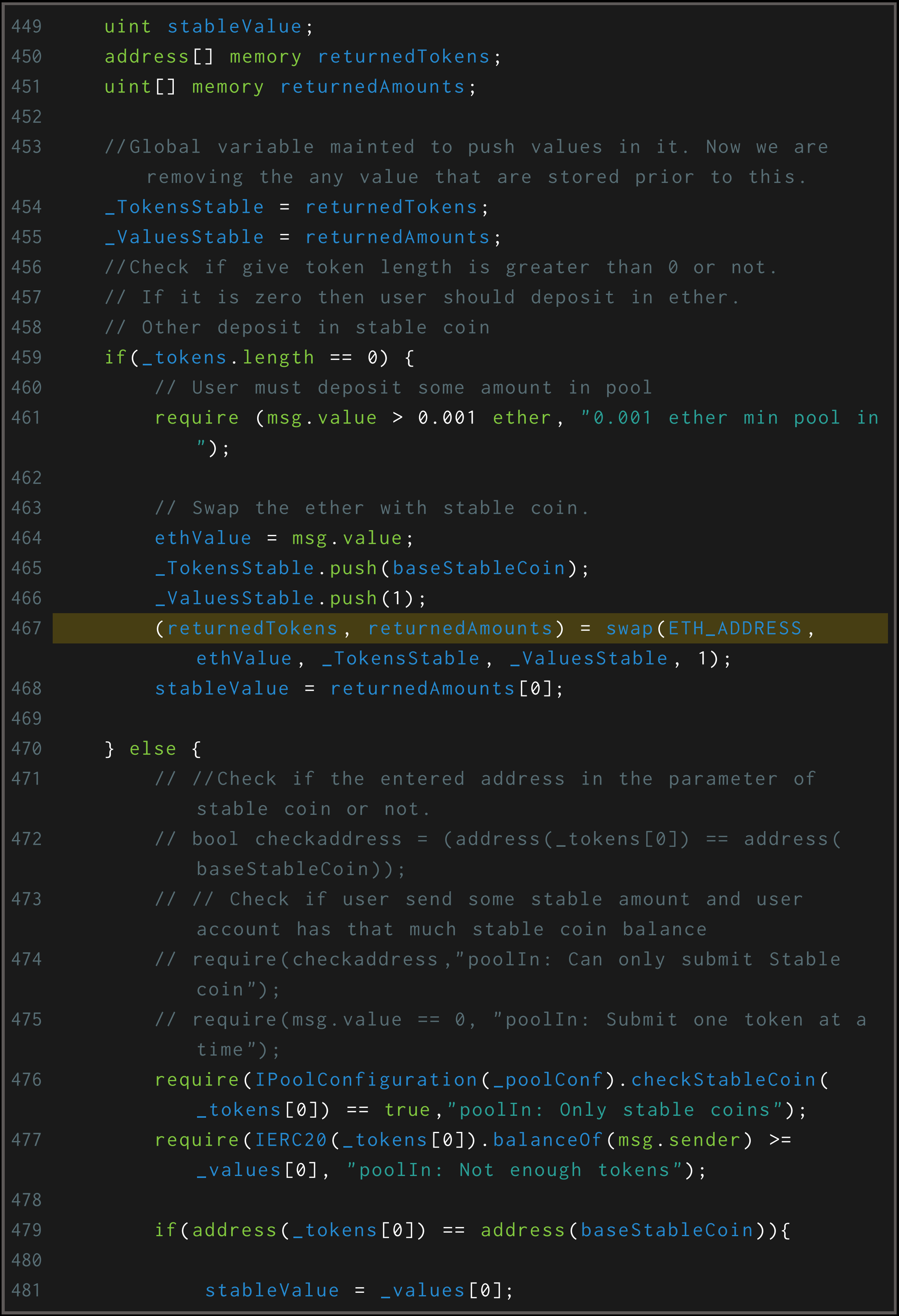
In the contracts poolv1.sol, chef.sol and lm-pool.sol the check, effects, interactions pattern is not being followed in some functions and this could open an attack vector for reentrancy attacks or code inconsistencies.

Code Location:

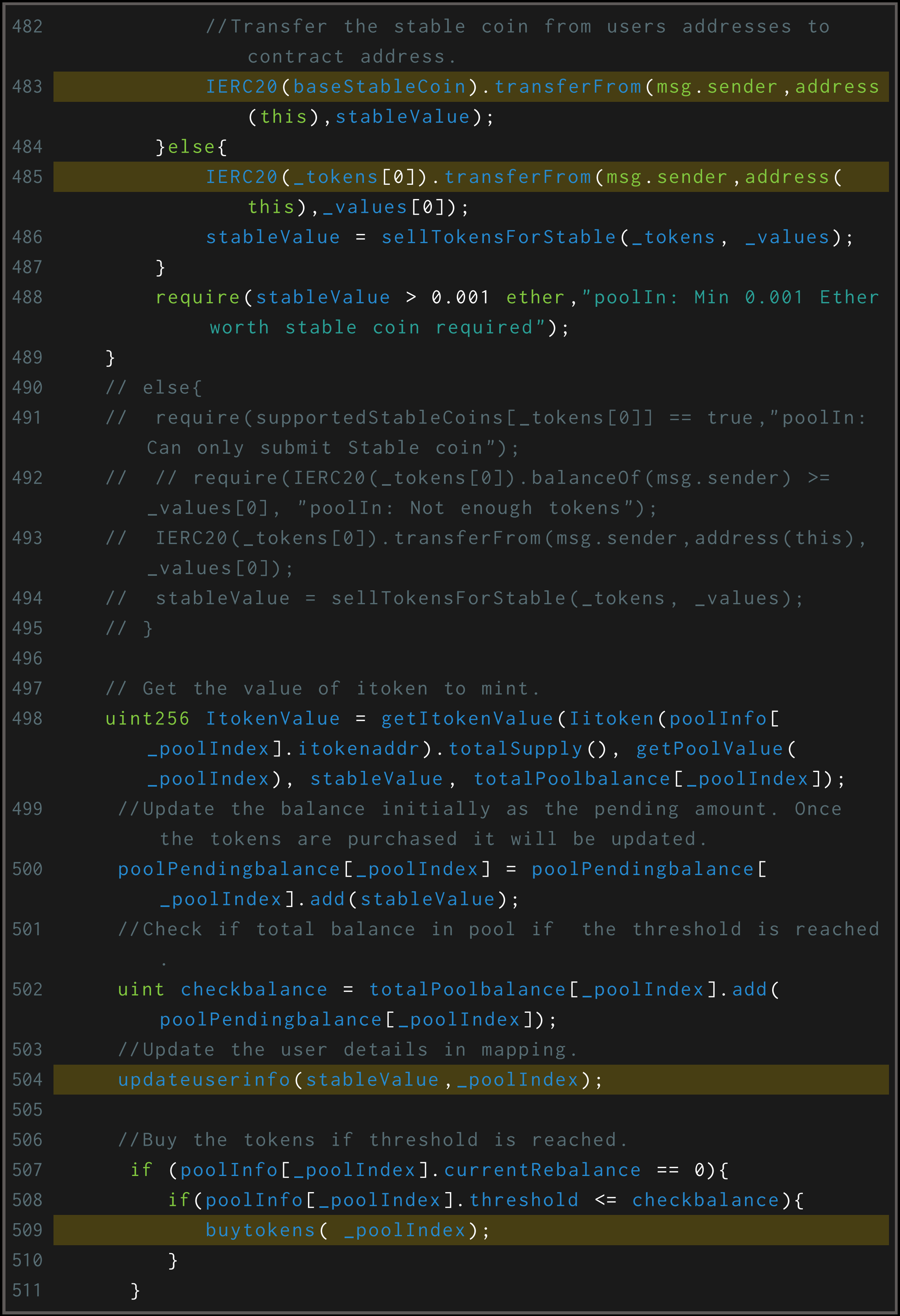
poolv1.sol

Listing 30: poolv1.sol (Lines 467,483,485,504,509,514)

```
434 function poolIn(address[] calldata _tokens, uint[] calldata _values, uint _poolIndex) external payable  {
435     // Require conditions to check if user is whitelisted or check the token configuration which user is depositing
436     // Only stable coin and Ether can be used in the initial stages.
437     require(poolUserInfo[_poolIndex][msg.sender].isenabled, "poolIn: Only whitelisted user");
438     require(_poolIndex<poolInfo.length, "poolIn: Invalid Pool Index");
439     require(_tokens.length <2 && _values.length<2, "poolIn: Only one token allowed");
440     // Check if is the first deposit or user already deposit before this. It will be used to calculate early exit fees
441     if(!existingUser[msg.sender][_poolIndex]){
442         existingUser[msg.sender][_poolIndex] = true;
443         initalDeposit[msg.sender][_poolIndex] = block.number;
444     }
445
446     // Variable that are used internally for logic/calling other functions.
447     uint ethValue;
448     uint fees;
```



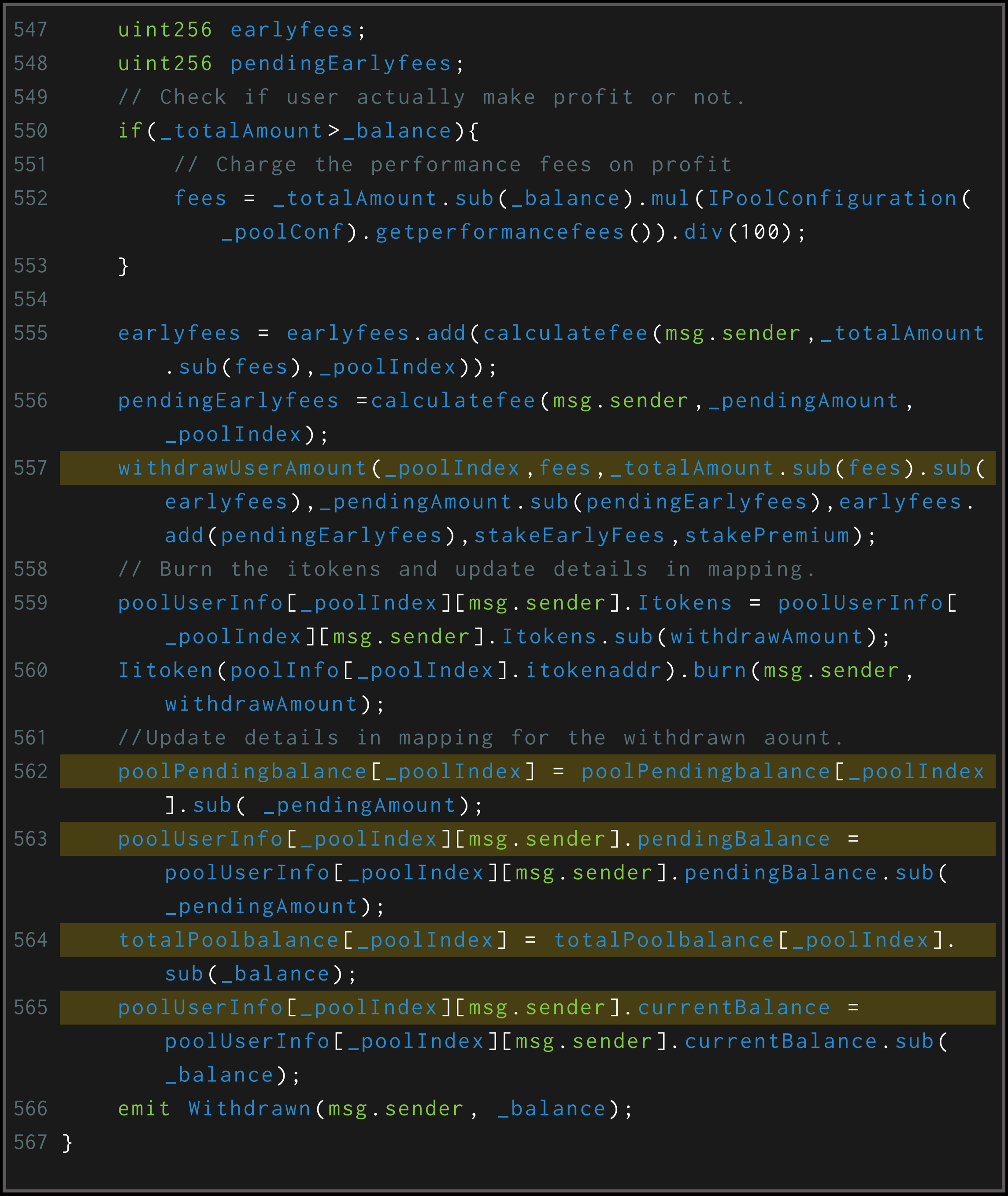
```
    uint stableValue;
    address[] memory returnedTokens;
    uint[] memory returnedAmounts;

    //Global variable mainted to push values in it. Now we are removing the any value that are stored prior to this.
    _TokensStable = returnedTokens;
    _ValuesStable = returnedAmounts;
    //Check if give token length is greater than 0 or not.
    // If it is zero then user should deposit in ether.
    // Other deposit in stable coin
    if(_tokens.length == 0) {
        // User must deposit some amount in pool
        require (msg.value > 0.001 ether, "0.001 ether min pool in");

        // Swap the ether with stable coin.
        ethValue = msg.value;
        _TokensStable.push(baseStableCoin);
        _ValuesStable.push(1);
        (returnedTokens, returnedAmounts) = swap(ETH_ADDRESS, ethValue, _TokensStable, _ValuesStable, 1);
        stableValue = returnedAmounts[0];

    } else {
        // //Check if the entered address in the parameter of stable coin or not.
        // bool checkaddress = (address(_tokens[0]) == address(baseStableCoin));
        // // Check if user send some stable amount and user account has that much stable coin balance
        // require(checkaddress,"poolIn: Can only submit Stable coin");
        // require(msg.value == 0, "poolIn: Submit one token at a time");
        require(IPoolConfiguration(_poolConf).checkStableCoin(_tokens[0]) == true,"poolIn: Only stable coins");
        require(IERC20(_tokens[0]).balanceOf(msg.sender) >= _values[0], "poolIn: Not enough tokens");

        if(address(_tokens[0]) == address(baseStableCoin)){

            stableValue = _values[0];
```



```
            //Transfer the stable coin from users addresses to contract address.
            IERC20(baseStableCoin).transferFrom(msg.sender,address(this),stableValue);
        }else{
            IERC20(_tokens[0]).transferFrom(msg.sender,address(this),_values[0]);
            stableValue = sellTokensForStable(_tokens, _values);
        }
        require(stableValue > 0.001 ether,"poolIn: Min 0.001 Ether worth stable coin required");
    }
    // else{
    //  require(supportedStableCoins[_tokens[0]] == true,"poolIn: Can only submit Stable coin");
    //  // require(IERC20(_tokens[0]).balanceOf(msg.sender) >= _values[0], "poolIn: Not enough tokens");
    //  IERC20(_tokens[0]).transferFrom(msg.sender,address(this), _values[0]);
    //  stableValue = sellTokensForStable(_tokens, _values);
    // }

    // Get the value of itoken to mint.
    uint256 ItokenValue = getItokenValue(Iitoken(poolInfo[_poolIndex].itokenaddr).totalSupply(), getPoolValue(_poolIndex), stableValue, totalPoolbalance[_poolIndex]);
     //Update the balance initially as the pending amount. Once the tokens are purchased it will be updated.
     poolPendingbalance[_poolIndex] = poolPendingbalance[_poolIndex].add(stableValue);
     //Check if total balance in pool if  the threshold is reached.
     uint checkbalance = totalPoolbalance[_poolIndex].add(poolPendingbalance[_poolIndex]);
     //Update the user details in mapping.
     updateuserinfo(stableValue,_poolIndex);

     //Buy the tokens if threshold is reached.
      if (poolInfo[_poolIndex].currentRebalance == 0){
         if(poolInfo[_poolIndex].threshold <= checkbalance){
            buytokens( _poolIndex);
         }
      }
    // poolOutstandingValue[_poolIndex] =  poolOutstandingValue[_poolIndex].add();
    // Again update details after tokens are bought.
    updateuserinfo(0,_poolIndex);
    //Mint new itokens and store details in mapping.
    poolUserInfo[_poolIndex][msg.sender].Itokens = poolUserInfo[_poolIndex][msg.sender].Itokens.add(ItokenValue);
    Iitoken(poolInfo[_poolIndex].itokenaddr).mint(msg.sender, ItokenValue);
}
```

Listing 31: poolv1.sol (Lines 557,562,563,564,565)

```
function withdraw(uint _poolIndex, bool stakeEarlyFees,bool stakePremium, uint withdrawAmount) external {
    require(_poolIndex<poolInfo.length, "Invalid Pool Index");
    require(Iitoken(poolInfo[_poolIndex].itokenaddr).balanceOf(msg.sender)>=withdrawAmount, "PoolV1: Not enough Itoken for Withdraw");
    // Update user info before withdrawal.
    updateuserinfo(0,_poolIndex);
    // Get the user share on the pool
    uint userShare = poolUserInfo[_poolIndex][msg.sender].currentBalance.add(poolUserInfo[_poolIndex][msg.sender].pendingBalance).mul(withdrawAmount).div(poolUserInfo[_poolIndex][msg.sender].Itokens);
    uint _balance;
    uint _pendingAmount;

    // Check if withdrawn amount is greater than pending amount. It will use the pending stable balance after that it will
    if(userShare>poolUserInfo[_poolIndex][msg.sender].pendingBalance){
        _balance = userShare.sub(poolUserInfo[_poolIndex][msg.sender].pendingBalance);
        _pendingAmount = poolUserInfo[_poolIndex][msg.sender].pendingBalance;
    }else{
        _pendingAmount = userShare;
    }
    // Call the functions to sell the tokens and recieve stable based on the user share in that pool
    uint256 _totalAmount = withdrawTokens(_poolIndex,_balance);
    uint fees;
```



```
    uint256 earlyfees;
    uint256 pendingEarlyfees;
    // Check if user actually make profit or not.
    if(_totalAmount>_balance){
        // Charge the performance fees on profit
        fees = _totalAmount.sub(_balance).mul(IPoolConfiguration(_poolConf).getperformancefees()).div(100);
    }

    earlyfees = earlyfees.add(calculatefee(msg.sender,_totalAmount.sub(fees),_poolIndex));
    pendingEarlyfees =calculatefee(msg.sender,_pendingAmount,_poolIndex);
    withdrawUserAmount(_poolIndex,fees,_totalAmount.sub(fees).sub(earlyfees),_pendingAmount.sub(pendingEarlyfees),earlyfees.add(pendingEarlyfees),stakeEarlyFees,stakePremium);
    // Burn the itokens and update details in mapping.
    poolUserInfo[_poolIndex][msg.sender].Itokens = poolUserInfo[_poolIndex][msg.sender].Itokens.sub(withdrawAmount);
    Iitoken(poolInfo[_poolIndex].itokenaddr).burn(msg.sender,withdrawAmount);
    //Update details in mapping for the withdrawn aount.
    poolPendingbalance[_poolIndex] = poolPendingbalance[_poolIndex].sub( _pendingAmount);
    poolUserInfo[_poolIndex][msg.sender].pendingBalance = poolUserInfo[_poolIndex][msg.sender].pendingBalance.sub(_pendingAmount);
    totalPoolbalance[_poolIndex] = totalPoolbalance[_poolIndex].sub(_balance);
    poolUserInfo[_poolIndex][msg.sender].currentBalance = poolUserInfo[_poolIndex][msg.sender].currentBalance.sub(_balance);
    emit Withdrawn(msg.sender, _balance);
}
```

Other functions also affected in poolv1.sol: withdrawTokens(), updatePool(), rebalance() and buytokens().

chef.sol

Listing 32: chef.sol (Lines 443,458,459,460,461,462,465,467)

```
428 function deposit(
429     uint256 _pid,
430     uint256 _amount,
431     uint256 vault
432 ) external {
433     require(vaultList[vault] == true, "no vault");
434     PoolInfo storage pool = poolInfo[_pid];
435     // This function is called for updating the total reward value
            which user is getting through block rewards
436     updateBlockReward(_pid, msg.sender);
437     UserInfo storage user = userInfo[_pid][msg.sender];
438     // This function is called to keep record of who is staking
           the tokens on the chef contract with pool id.
439     addUserAddress(msg.sender, _pid);
440     if (_amount > 0) {
441         // Here if entered amount is greater than 0 then that
               amount would be transferred from user account to
442         // chef contract
443         pool.lpToken.safeTransferFrom(
444             address(msg.sender),
445             address(this),
446             _amount
447         );
448         user.amount = user.amount.add(_amount);
449     }
450     // Updating staking score structure after staking the tokens
451     userStakingTrack[_pid][msg.sender] = userStakingTrack[_pid][
           msg.sender]
452         .add(1);
453     // Set the id of user staking info.
454     uint256 userstakeid = userStakingTrack[_pid][msg.sender];
455     // Fetch the stakeInfo which saved on stake id.
456     StakeInfo storage staker = stakeInfo[_pid][msg.sender][
           userstakeid];
457     // Here sets the below values in the object.
458     staker.amount = _amount;
459     staker.totalAmount = user.amount;
460     staker.timestamp = block.timestamp;
461     staker.vault = vault;
462     staker.deposit = true;
463
```

```
    //user timestamp
    user.timestamp = block.timestamp;
    // update hishest staker array
    addHighestStakedUser(_pid, user.amount, msg.sender);
    emit Deposit(msg.sender, _pid, _amount);
}
```

Listing 33: chef.sol (Lines 1364,1365)

```
1353 function withdrawASTRReward(uint256 _pid, bool _withStake) public
        {
1354   // bool isValid = Dao(daoAddress).getVotingStatus(msg.sender);
1355   // require(isValid==true, "should vote active proposal");
1356
1357   // Update the block reward for the current user.
1358   updateBlockReward(_pid, msg.sender);
1359   UserInfo storage currentUser = userInfo[_pid][msg.sender];
1360   if (_withStake) {
1361       // If user choses to withdraw the ASTRA with staking it in to
               astra.
1362       uint256 _amount = currentUser.totalReward;
1363       // Called this function for staking the ASTRA rewards in
              astra pool.
1364       _stakeASTRReward(msg.sender, ASTRPoolId, _amount);
1365       updateClaimedReward(currentUser, _amount);
1366   } else {
1367       // Else we will slash some fee and send the amount to user
              account.
1368       uint256 dayInSecond = 86400;
1369       uint256 dayCount =
1370           (block.timestamp.sub(currentUser.timestamp)).div(
                  dayInSecond);
1371       if (dayCount >= 90) {
1372           dayCount = 90;
1373       }
1374       // Called this function for slashing fee from reward if claim
               is happend with in 90 days.
1375       slashExitFee(currentUser, _pid, dayCount);
1376   }
1377   // Updating the total reward to 0 in UserInfo object.
1378   currentUser.totalReward = 0;
1379 }
```

Other functions also affected in chef.sol: depositFromDAA() and withdrawASTRReward().

lm-pool.sol

Listing 34: lm-pool.sol (Lines 261,276,277,278,279,280)

```
250 function deposit(
251     uint256 _pid,
252     uint256 _amount,
253     uint256 vault
254 ) external {
255     require(vaultList[vault] == true, "no vault");
256     PoolInfo storage pool = poolInfo[_pid];
257     updateBlockReward(_pid);
258     UserInfo storage user = userInfo[_pid][msg.sender];
259     addUserAddress(_pid);
260     if (_amount > 0) {
261         pool.lpToken.safeTransferFrom(
262             address(msg.sender),
263             address(this),
264             _amount
265         );
266         user.amount = user.amount.add(_amount);
267     }
268     // Updating staking score structure after staking the tokens
269     userStakingTrack[_pid][msg.sender] = userStakingTrack[_pid][
        msg.sender]
270         .add(1);
271     // Set the id of user staking info.
272     uint256 userstakeid = userStakingTrack[_pid][msg.sender];
273     // Fetch the stakeInfo which saved on stake id.
274     StakeInfo storage staker = stakeInfo[_pid][msg.sender][
        userstakeid];
275     // Here sets the below values in the object.
276     staker.amount = _amount;
277     staker.totalAmount = user.amount;
278     staker.timestamp = block.timestamp;
279     staker.vault = vault;
280     staker.deposit = true;
281
282     //user timestamp
283     user.timestamp = block.timestamp;
284     emit Deposit(msg.sender, _pid, _amount);
285 }
```

Listing 35: lm-pool.sol (Lines 788,789)

```
780 function withdrawASTRReward(uint256 _pid, bool _withStake) public
       {
781     // Update the block reward for the current user.
782     updateBlockReward(_pid);
783     UserInfo storage currentUser = userInfo[_pid][msg.sender];
784     if (_withStake) {
785         // If user choses to withdraw the ASTRA with staking it in
                to astra.
786         uint256 _amount = currentUser.totalReward;
787         // Called this function for staking the ASTRA rewards in
               astra pool.
788         stakeASTRReward(Chef(chefaddr).ASTRPoolId(), _amount);
789         updateClaimedReward(currentUser, _amount);
790     } else {
791         // Else we will slash some fee and send the amount to user
                account.
792         uint256 dayInSecond = 86400;
793         uint256 dayCount =
794             (block.timestamp.sub(currentUser.timestamp)).div(
                   dayInSecond);
795         if (dayCount >= 90) {
796             dayCount = 90;
797         }
798         // Called this function for slashing fee from reward if
               claim is happend with in 90 days.
799         slashExitFee(currentUser, _pid, dayCount);
800     }
801     // Updating the total reward to 0 in UserInfo object.
802     currentUser.totalReward = 0;
803 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Follow the check, effects, interactions pattern.

Remediation Plan:

SOLVED: AstraDAO Team added the nonReentrant modifier in all the external/public functions affected to prevent reentrancy.

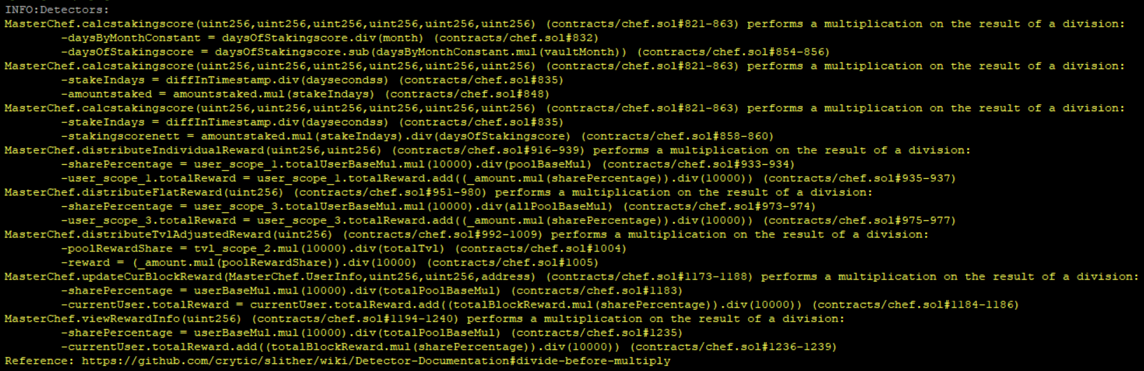
3.9 (HAL-09) DIVIDE BEFORE MULTIPLY - LOW

Description:

Solidity integer division might truncate. As a result, performing multiplication before division might reduce precision. As the contracts chef.sol and lm-pool.sol handles the payout bonuses, the voting power. . . the sensitivity of precision of the mathematical operations in these contracts should be considered critical.

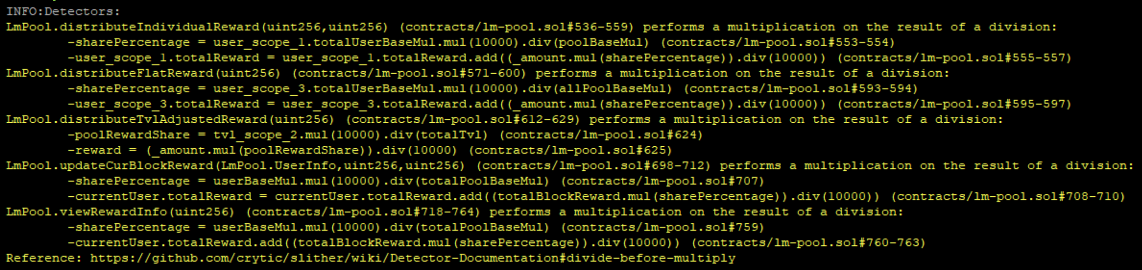
Code Location:

chef.sol



```
INFO:Detectors:
MasterChef.calcstakingscore(uint256,uint256,uint256,uint256,uint256,uint256) (contracts/chef.sol#821-863) performs a multiplication on the result of a division:
        -daysByMonthConstant = daysOfStakingscore.div(month) (contracts/chef.sol#832)
        -daysOfStakingscore = daysOfStakingscore.sub(daysByMonthConstant.mul(vaultMonth)) (contracts/chef.sol#854-856)
MasterChef.calcstakingscore(uint256,uint256,uint256,uint256,uint256,uint256) (contracts/chef.sol#821-863) performs a multiplication on the result of a division:
        -stakeIndays = diffInTimestamp.div(daysecondss) (contracts/chef.sol#835)
        -amountstaked = amountstaked.mul(stakeIndays) (contracts/chef.sol#848)
MasterChef.calcstakingscore(uint256,uint256,uint256,uint256,uint256,uint256) (contracts/chef.sol#821-863) performs a multiplication on the result of a division:
        -stakeIndays = diffInTimestamp.div(daysecondss) (contracts/chef.sol#835)
        -stakingscorenett = amountstaked.mul(stakeIndays).div(daysOfStakingscore) (contracts/chef.sol#858-860)
MasterChef.distributeIndividualReward(uint256,uint256) (contracts/chef.sol#916-939) performs a multiplication on the result of a division:
        -sharePercentage = user_scope_1.totalUserBaseMul.mul(10000).div(poolBaseMul) (contracts/chef.sol#933-934)
        -user_scope_1.totalReward = user_scope_1.totalReward.add((_amount.mul(sharePercentage)).div(10000)) (contracts/chef.sol#935-937)
MasterChef.distributeFlatReward(uint256) (contracts/chef.sol#951-980) performs a multiplication on the result of a division:
        -sharePercentage = user_scope_3.totalUserBaseMul.mul(10000).div(allPoolBaseMul) (contracts/chef.sol#973-974)
        -user_scope_3.totalReward = user_scope_3.totalReward.add((_amount.mul(sharePercentage)).div(10000)) (contracts/chef.sol#975-977)
MasterChef.distributeTvlAdjustedReward(uint256) (contracts/chef.sol#992-1009) performs a multiplication on the result of a division:
        -poolRewardShare = tvl_scope_2.mul(10000).div(totalTvl) (contracts/chef.sol#1004)
        -reward = (_amount.mul(poolRewardShare)).div(10000) (contracts/chef.sol#1005)
MasterChef.updateCurBlockReward(MasterChef.UserInfo,uint256,uint256,address) (contracts/chef.sol#1173-1188) performs a multiplication on the result of a division:
        -sharePercentage = userBaseMul.mul(10000).div(totalPoolBaseMul) (contracts/chef.sol#1183)
        -currentUser.totalReward = currentUser.totalReward.add((totalBlockReward.mul(sharePercentage)).div(10000)) (contracts/chef.sol#1184-1186)
MasterChef.viewRewardInfo(uint256) (contracts/chef.sol#1194-1240) performs a multiplication on the result of a division:
        -sharePercentage = userBaseMul.mul(10000).div(totalPoolBaseMul) (contracts/chef.sol#1235)
        -currentUser.totalReward.add((totalBlockReward.mul(sharePercentage)).div(10000)) (contracts/chef.sol#1236-1239)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
```

lm-pool.sol



```
INFO:Detectors:
LmPool.distributeIndividualReward(uint256,uint256) (contracts/lm-pool.sol#536-559) performs a multiplication on the result of a division:
        -sharePercentage = user_scope_1.totalUserBaseMul.mul(10000).div(poolBaseMul) (contracts/lm-pool.sol#553-554)
        -user_scope_1.totalReward = user_scope_1.totalReward.add((_amount.mul(sharePercentage)).div(10000)) (contracts/lm-pool.sol#555-557)
LmPool.distributeFlatReward(uint256) (contracts/lm-pool.sol#571-600) performs a multiplication on the result of a division:
        -sharePercentage = user_scope_3.totalUserBaseMul.mul(10000).div(allPoolBaseMul) (contracts/lm-pool.sol#593-594)
        -user_scope_3.totalReward = user_scope_3.totalReward.add((_amount.mul(sharePercentage)).div(10000)) (contracts/lm-pool.sol#595-597)
LmPool.distributeTvlAdjustedReward(uint256) (contracts/lm-pool.sol#612-629) performs a multiplication on the result of a division:
        -poolRewardShare = tvl_scope_2.mul(10000).div(totalTvl) (contracts/lm-pool.sol#624)
        -reward = (_amount.mul(poolRewardShare)).div(10000) (contracts/lm-pool.sol#625)
LmPool.updateCurBlockReward(LmPool.UserInfo,uint256,uint256) (contracts/lm-pool.sol#698-712) performs a multiplication on the result of a division:
        -sharePercentage = userBaseMul.mul(10000).div(totalPoolBaseMul) (contracts/lm-pool.sol#707)
        -currentUser.totalReward = currentUser.totalReward.add((totalBlockReward.mul(sharePercentage)).div(10000)) (contracts/lm-pool.sol#708-710)
LmPool.viewRewardInfo(uint256) (contracts/lm-pool.sol#718-764) performs a multiplication on the result of a division:
        -sharePercentage = userBaseMul.mul(10000).div(totalPoolBaseMul) (contracts/lm-pool.sol#759)
        -currentUser.totalReward.add((totalBlockReward.mul(sharePercentage)).div(10000)) (contracts/lm-pool.sol#760-763)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider ordering multiplication before division.

Remediation Plan:

RISK ACCEPTED: AstraDAO Team accepts this risk.

3.10 (HAL-10) TAUTOLOGY EXPRESSIONS - LOW

Description:

In the contracts chef.sol and lm-pool.sol a tautology expression has been detected. Such expressions are of no use since they always evaluate true/false regardless of the context they are used in.

Code Location:

chef.sol

Listing 36: chef.sol (Lines 1456,1457,1458)

```
1447 function slashExitFee(
1448     UserInfo storage currentUser,
1449     uint256 _pid,
1450     uint256 dayCount
1451 ) private {
1452     uint256 totalReward = currentUser.totalReward;
1453     uint256 sfr = uint256(90).sub(dayCount);
1454     // Here fee is calculated on the basis of how days is left in 90 days.
1455     uint256 fee = totalReward.mul(sfr).div(100);
1456     if (fee < 0) {
1457         fee = 0;
1458     }
1459     // Claimable reward is calculated by substracting the fee from total reward.
1460     uint256 claimableReward = totalReward.sub(fee);
1461     if (claimableReward > 0) {
1462         safeASTRTransfer(msg.sender, claimableReward);
1463         currentUser.totalReward = 0;
1464     }
1465     // Deducted fee would be distribute as reward to the same pool user as individual reward
1466     // with reward multiplier logic.
1467     distributeIndividualReward(_pid, fee);
1468     updateClaimedReward(currentUser, claimableReward);
1469 }
```

lm-pool.sol

Listing 37: chef.sol (Lines 842,843,844)

```
833 function slashExitFee(
834     UserInfo storage currentUser,
835     uint256 _pid,
836     uint256 dayCount
837 ) private {
838     uint256 totalReward = currentUser.totalReward;
839     uint256 sfr = uint256(90).sub(dayCount);
840     // Here fee is calculated on the basis of how days is left in
           90 days.
841     uint256 fee = totalReward.mul(sfr).div(100);
842     if (fee < 0) {
843         fee = 0;
844     }
845     // Claimable reward is calculated by substracting the fee from
            total reward.
846     uint256 claimableReward = totalReward.sub(fee);
847     if (claimableReward > 0) {
848         safeASTRTransfer(msg.sender, claimableReward);
849         currentUser.totalReward = 0;
850     }
851     // Deducted fee would be distribute as reward to the same pool
            user as individual reward
852     // with reward multiplier logic.
853     distributeIndividualReward(_pid, fee);
854     updateClaimedReward(currentUser, claimableReward);
855 }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Checking if a uint256-type value is lower than zero is not necessary: uint256 is in range $\langle 0, 2^{256} - 1 \rangle$

Remediation Plan:

SOLVED: AstraDAO Team removed all tautology expressions.

3.11 (HAL-11) USE OF INLINE ASSEMBLY - INFORMATIONAL

Description:

Inline assembly is a way to access the Ethereum Virtual Machine at a low level. This discards several important safety features in Solidity. Inline assembly is used in the imported strings library in oracle.sol contract and is also used in the constructor and in a function of governance.sol contract:

Code Location:

oracle.sol

Listing 38

```
version-5/oracle.sol:27: assembly {
version-5/oracle.sol:63: assembly {
version-5/oracle.sol:72: assembly {
version-5/oracle.sol:87: assembly { retptr := add(ret, 32) }
version-5/oracle.sol:105: assembly { needledata := and(mload(needleptr), mask) }
version-5/oracle.sol:109: assembly { ptrdata := and(mload(ptr), mask) }
version-5/oracle.sol:115: assembly { ptrdata := and(mload(ptr), mask) }
version-5/oracle.sol:121: assembly { hash := keccak256(needleptr, needlelen) }
version-5/oracle.sol:125: assembly { testHash := keccak256(ptr, needlelen) }
```

governance.sol

Listing 39

```
version-5/governance.sol:55: assembly { cs := extcodesize(address) }
version-5/governance.sol:659: assembly { chainId := chainid() }
```

Risk Level:

Likelihood - 1
Impact - 2

Recommendation:

When possible, do not use inline assembly because it is a manner to access to the EVM (Ethereum Virtual Machine) at a low level. An attacker could bypass many important safety features of Solidity.

Remediation Plan:

ACKNOWLEDGED: AstraDAO Team acknowledges this issue, as inline assembly is used by referenced libraries like initializable and string.

3.12 (HAL-12) WITHDRAW COOLDOWN PERIOD CAN BE BYPASSED - INFORMATIONAL

Description:

In the chef.sol and lm-pool.sol contracts, stakers willing to withdraw tokens from the staking pool will need to go through 7 days of cooldown period. After 7 days, if the user fails to confirm the unstake transaction in the 24h time window, the cooldown period will be reset.

By following these steps a user can bypass the cooldown period:

1. Deposit 1 token
2. Ask for a withdraw()
3. Wait 7 days
4. Do a 2nd deposit of xyz tokens
5. Call withdraw again retrieving the first token deposited plus the xyz tokens deposited in the 2nd deposit

Halborn advices that this could open an attack vector and be abused using flash loans. At this moment, this can not be abused as for example the voting power can not be increased immediately after a deposit, but developers should keep this threat in mind for future updates.

Code Location:

chef.sol

Listing 40: chef.sol

```
function withdraw(uint256 _pid, bool _withStake) external {
    UserInfo storage user = userInfo[_pid][msg.sender];
    uint256 _amount = viewEligibleAmount(_pid, msg.sender);
    require(_amount > 0, "withdraw: not good");
    //Instead of transferring to a standard staking vault, Astra
        tokens can be locked (meaning that staker forfeits the
        right to unstake them for a fixed period of time). There
```

```
         are following lockups vaults: 6,9 and 12 months.
    if (user.cooldown == false) {
        user.cooldown = true;
        user.cooldowntimestamp = block.timestamp;
        return;
    } else {
        // Stakers willing to withdraw tokens from the staking pool will need to go through 7 days
        // of cool-down period. After 7 days, if the user fails to confirm the unstake transaction in the 24h time window, the cooldown period will be reset.
        if (
            block.timestamp > user.cooldowntimestamp.add(dayseconds.mul(8))
        ) {
            user.cooldown = true;
            user.cooldowntimestamp = block.timestamp;
            return;
        } else {
            require(user.cooldown == true, "withdraw: cooldown status");
            require(
                block.timestamp >=
                    user.cooldowntimestamp.add(dayseconds.mul(7)),
                "withdraw: cooldown period"
            );
            require(
                block.timestamp <=
                    user.cooldowntimestamp.add(dayseconds.mul(8)),
                "withdraw: open window"
            );
            // Calling withdraw function after all the validation like cooldown period, eligible amount etc.
            _withdraw(_pid, _withStake);
        }
    }
}
```

lm-pool.sol

Listing 41: lm-pool.sol

```
function withdraw(uint256 _pid, bool _withStake) external {
    UserInfo storage user = userInfo[_pid][msg.sender];
    uint256 _amount = viewEligibleAmount(_pid, msg.sender);
    require(_amount > 0, "withdraw: not good");
    //Instead of transferring to a standard staking vault, Astra tokens can be locked (meaning that staker forfeits the right to unstake them for a fixed period of time). There are following lockups vaults: 6,9 and 12 months.
    if (user.cooldown == false) {
        user.cooldown = true;
        user.cooldowntimestamp = block.timestamp;
        return;
    } else {
        // Stakers willing to withdraw tokens from the staking pool will need to go through 7 days
        // of cool-down period. After 7 days, if the user fails to confirm the unstake transaction in the 24h time window, the cooldown period will be reset.
        if (
            block.timestamp > user.cooldowntimestamp.add(daysecends.mul(8))
        ) {
            user.cooldown = true;
            user.cooldowntimestamp = block.timestamp;
            return;
        } else {
            require(user.cooldown == true, "withdraw: cooldown status");
            require(
                block.timestamp >=
                    user.cooldowntimestamp.add(dayseconds.mul(7)),
                "withdraw: cooldown period"
            );
            require(
                block.timestamp <=
                    user.cooldowntimestamp.add(dayseconds.mul(8)),
                "withdraw: open window"
            );
            // Calling withdraw function after all the validation like cooldown period, eligible amount etc.
            _withdraw(_pid, _withStake);
        }
    }
}
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to redesign the withdraw() function so the cooldown period gets reset every time a deposit is done. Also, keep in mind that this bypass can be paired with flash loans for future code updates.

Remediation Plan:

ACKNOWLEDGED: As in HAL02 - SLASHING FEES/REDEPOSITS INCORRECT BEHAVIOUR , AstraDAO Team accepts this risk as the fix would add too much complexity into the smart contracts. AstraDAO Team will educate their users and mention this edge case in their whitepaper so every one is aware of this issue. AstraDAO Team will consider implementing a fix in the Phase 2.

3.13 (HAL-13) TYPO IN FUNCTION AND VARIABLE - INFORMATIONAL

Description:

In the contract poolv1.sol there are two typos, one in a state variable and another one in a function name.

Code Location:

poolv1.sol

Listing 42: poolv1.sol

```
mapping(address =>mapping (uint256 => uint256)) public
    initalDeposit;
```

initalDeposit should be named initialDeposit.

Listing 43: poolv1.sol

```
function calculateMinimumRetrun(uint _amount) internal view
    returns (uint){
    // This will get the slippage rate from configuration contract
         and calculate how much amount user can get after slippage.
    uint256 sliprate= IPoolConfiguration(_poolConf).
        getslippagerate();
    uint rate = _amount.mul(sliprate).div(100);
    // Return amount after calculating slippage
    return _amount.sub(rate);
}
```

calculateMinimumRetrun should be named calculateMinimumReturn.

Risk Level:

Likelihood - 1

Impact - 1

Recommendation:

Rename the variable and the function name.

Remediation Plan:

SOLVED: AstraDAO Team corrected the function name which is now called calculateMinimumReturn.

AUTOMATED TESTING

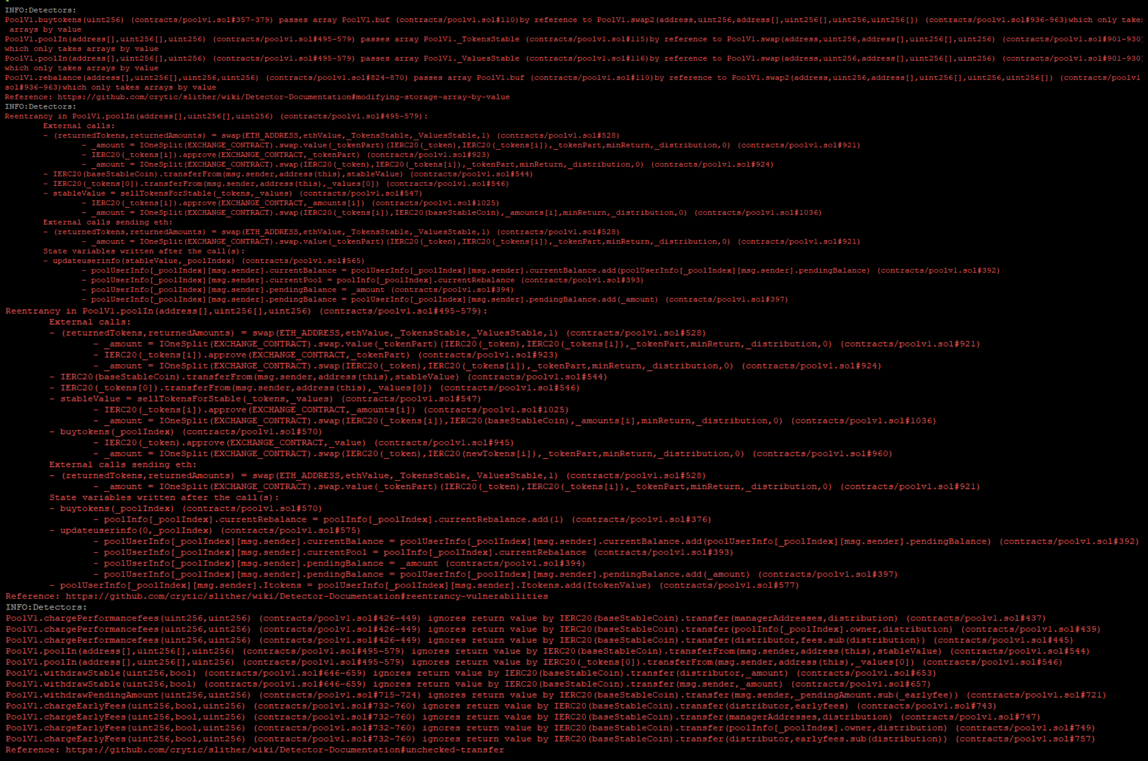
4.1 STATIC ANALYSIS REPORT

Description:

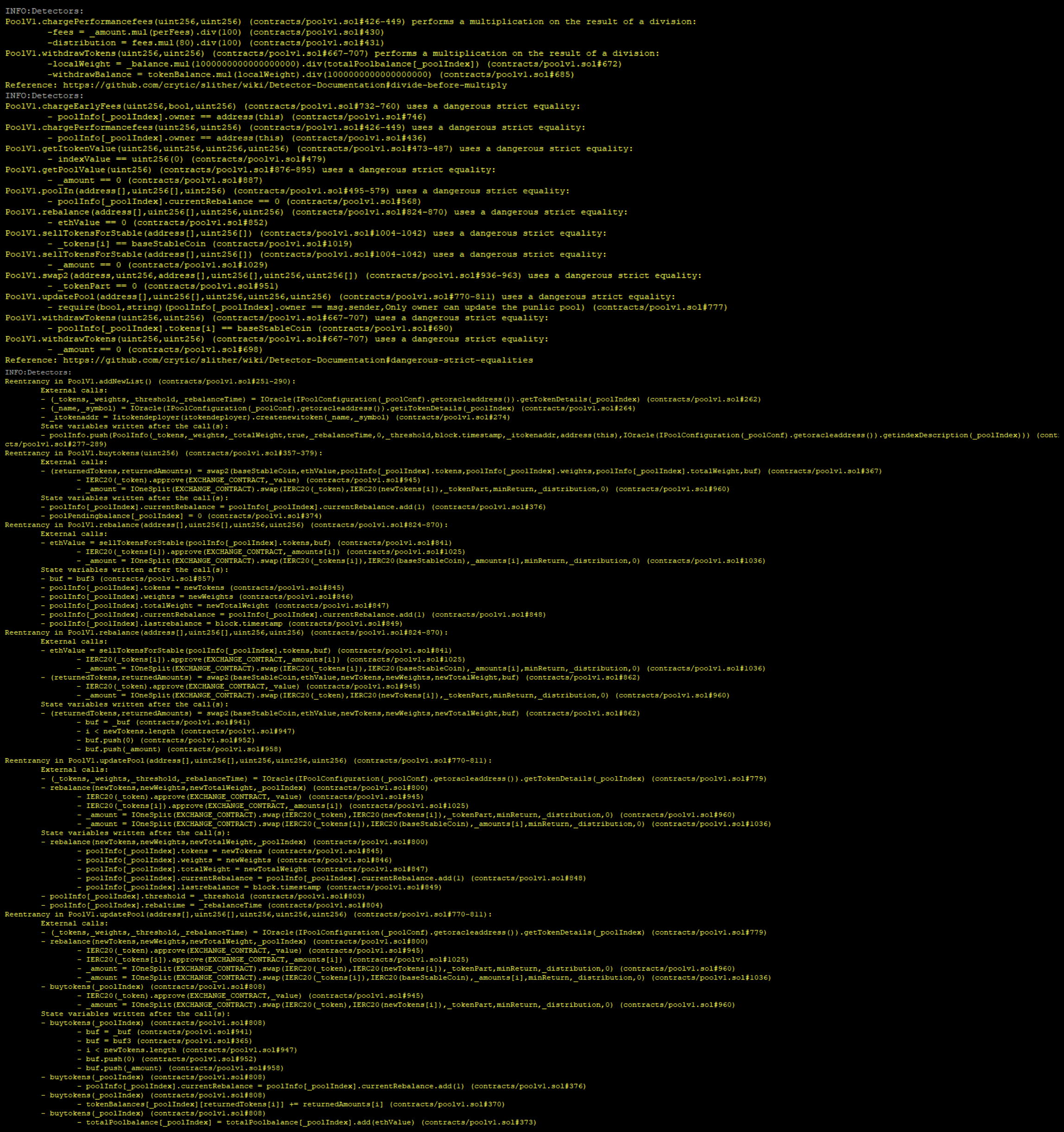
Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contracts. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their abi and binary formats, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Slither results:

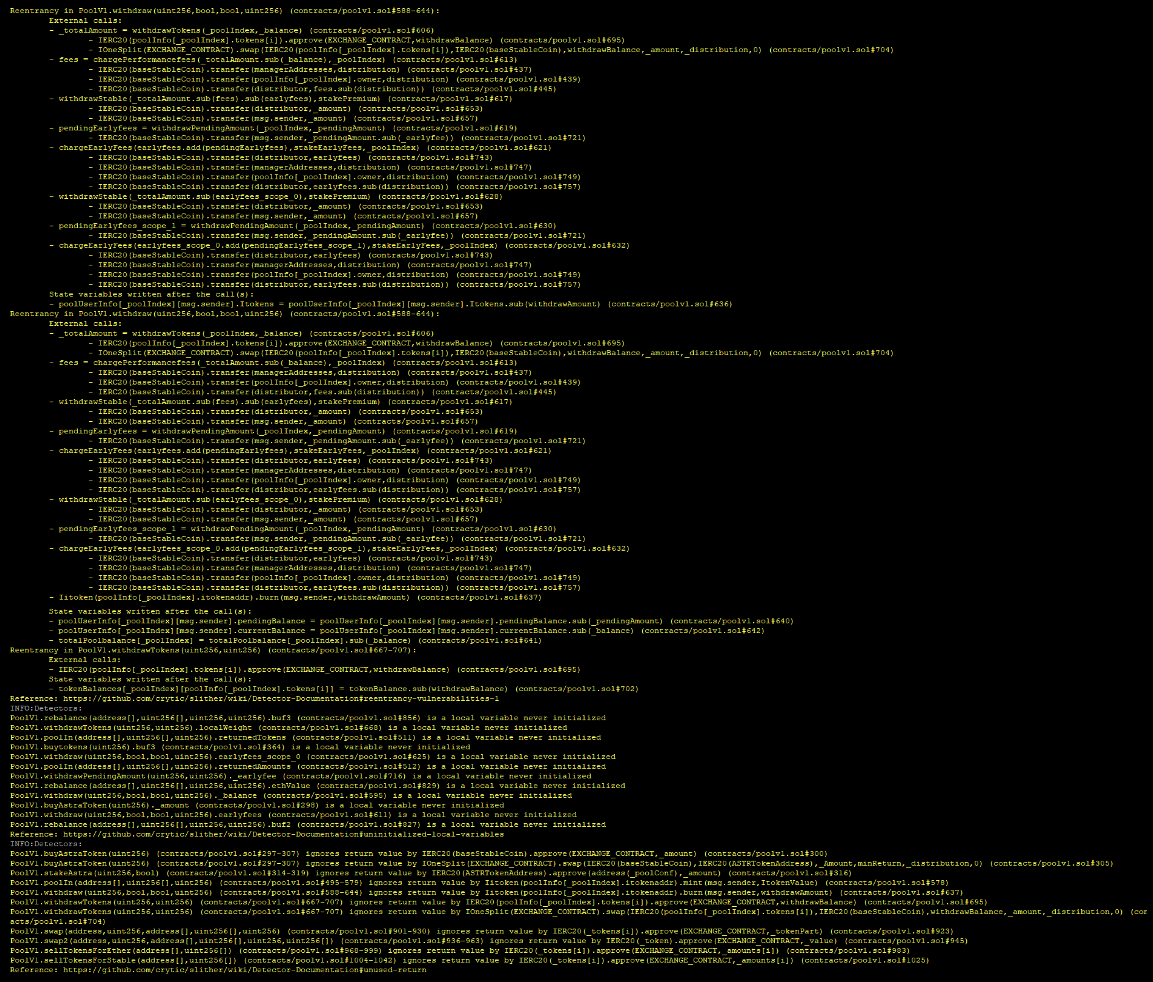
poolv1.sol



```
INFO:Detectors:
PoolV1.buytokens(uint256) (contracts/poolv1.sol#357-379) passes array PoolV1.buf (contracts/poolv1.sol#110)by reference to PoolV1.swap2(address,uint256,address[],uint256[],uint256,uint256[]) (contracts/poolv1.sol#936-963)which only takes
 arrays by value
PoolV1.poolIn(address[],uint256[],uint256) (contracts/poolv1.sol#495-579) passes array PoolV1._TokensStable (contracts/poolv1.sol#115)by reference to PoolV1.swap(address,uint256,address[],uint256[],uint256) (contracts/poolv1.sol#901-930)
which only takes arrays by value
PoolV1.poolIn(address[],uint256[],uint256) (contracts/poolv1.sol#495-579) passes array PoolV1._ValuesStable (contracts/poolv1.sol#116)by reference to PoolV1.swap(address,uint256,address[],uint256[],uint256) (contracts/poolv1.sol#901-930)
which only takes arrays by value
PoolV1.rebalance(address[],uint256[],uint256,uint256) (contracts/poolv1.sol#824-870) passes array PoolV1.buf (contracts/poolv1.sol#110)by reference to PoolV1.swap2(address,uint256,address[],uint256[],uint256,uint256[]) (contracts/poolv1
sol#936-963)which only takes arrays by value
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#modifying-storage-array-by-value
INFO:Detectors:
Reentrancy in PoolV1.poolIn(address[],uint256[],uint256) (contracts/poolv1.sol#495-579):
        External calls:
        - (returnedTokens,returnedAmounts) = swap(ETH_ADDRESS,ethValue,_TokensStable,_ValuesStable,1) (contracts/poolv1.sol#528)
                - _amount = IOneSplit(EXCHANGE_CONTRACT).swap.value(_tokenPart)(IERC20(_token),IERC20(_tokens[i]),_tokenPart,minReturn,_distribution,0) (contracts/poolv1.sol#921)
                - IERC20(_tokens[i]).approve(EXCHANGE_CONTRACT,_tokenPart) (contracts/poolv1.sol#923)
                - _amount = IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(_token),IERC20(_tokens[i]),_tokenPart,minReturn,_distribution,0) (contracts/poolv1.sol#924)
        - IERC20(baseStableCoin).transferFrom(msg.sender,address(this),stableValue) (contracts/poolv1.sol#544)
        - IERC20(_tokens[0]).transferFrom(msg.sender,address(this),_values[0]) (contracts/poolv1.sol#546)
        - stableValue = sellTokensForStable(_tokens,_values) (contracts/poolv1.sol#547)
                - IERC20(_tokens[i]).approve(EXCHANGE_CONTRACT,_amounts[i]) (contracts/poolv1.sol#1025)
                - _amount = IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(_tokens[i]),IERC20(baseStableCoin),_amounts[i],minReturn,_distribution,0) (contracts/poolv1.sol#1036)
        External calls sending eth:
        - (returnedTokens,returnedAmounts) = swap(ETH_ADDRESS,ethValue,_TokensStable,_ValuesStable,1) (contracts/poolv1.sol#528)
                - _amount = IOneSplit(EXCHANGE_CONTRACT).swap.value(_tokenPart)(IERC20(_token),IERC20(_tokens[i]),_tokenPart,minReturn,_distribution,0) (contracts/poolv1.sol#921)
        State variables written after the call(s):
        - updateuserinfo(stableValue,_poolIndex) (contracts/poolv1.sol#565)
                - poolUserInfo[_poolIndex][msg.sender].currentBalance = poolUserInfo[_poolIndex][msg.sender].currentBalance.add(poolUserInfo[_poolIndex][msg.sender].pendingBalance) (contracts/poolv1.sol#392)
                - poolUserInfo[_poolIndex][msg.sender].currentPool = poolInfo[_poolIndex].currentRebalance (contracts/poolv1.sol#393)
                - poolUserInfo[_poolIndex][msg.sender].pendingBalance = _amount (contracts/poolv1.sol#394)
                - poolUserInfo[_poolIndex][msg.sender].pendingBalance = poolUserInfo[_poolIndex][msg.sender].pendingBalance.add(_amount) (contracts/poolv1.sol#397)
Reentrancy in PoolV1.poolIn(address[],uint256[],uint256) (contracts/poolv1.sol#495-579):
        External calls:
        - (returnedTokens,returnedAmounts) = swap(ETH_ADDRESS,ethValue,_TokensStable,_ValuesStable,1) (contracts/poolv1.sol#528)
                - _amount = IOneSplit(EXCHANGE_CONTRACT).swap.value(_tokenPart)(IERC20(_token),IERC20(_tokens[i]),_tokenPart,minReturn,_distribution,0) (contracts/poolv1.sol#921)
                - IERC20(_tokens[i]).approve(EXCHANGE_CONTRACT,_tokenPart) (contracts/poolv1.sol#923)
                - _amount = IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(_token),IERC20(_tokens[i]),_tokenPart,minReturn,_distribution,0) (contracts/poolv1.sol#924)
        - IERC20(baseStableCoin).transferFrom(msg.sender,address(this),stableValue) (contracts/poolv1.sol#544)
        - IERC20(_tokens[0]).transferFrom(msg.sender,address(this),_values[0]) (contracts/poolv1.sol#546)
        - stableValue = sellTokensForStable(_tokens,_values) (contracts/poolv1.sol#547)
                - IERC20(_tokens[i]).approve(EXCHANGE_CONTRACT,_amounts[i]) (contracts/poolv1.sol#1025)
                - _amount = IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(_tokens[i]),IERC20(baseStableCoin),_amounts[i],minReturn,_distribution,0) (contracts/poolv1.sol#1036)
        - buytokens(_poolIndex) (contracts/poolv1.sol#570)
                - IERC20(_token).approve(EXCHANGE_CONTRACT,_value) (contracts/poolv1.sol#945)
                - _amount = IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(_token),IERC20(newTokens[i]),_tokenPart,minReturn,_distribution,0) (contracts/poolv1.sol#960)
        External calls sending eth:
        - (returnedTokens,returnedAmounts) = swap(ETH_ADDRESS,ethValue,_TokensStable,_ValuesStable,1) (contracts/poolv1.sol#528)
                - _amount = IOneSplit(EXCHANGE_CONTRACT).swap.value(_tokenPart)(IERC20(_token),IERC20(_tokens[i]),_tokenPart,minReturn,_distribution,0) (contracts/poolv1.sol#921)
        State variables written after the call(s):
        - buytokens(_poolIndex) (contracts/poolv1.sol#570)
                - poolInfo[_poolIndex].currentRebalance = poolInfo[_poolIndex].currentRebalance.add(1) (contracts/poolv1.sol#376)
        - updateuserinfo(0,_poolIndex) (contracts/poolv1.sol#575)
                - poolUserInfo[_poolIndex][msg.sender].currentBalance = poolUserInfo[_poolIndex][msg.sender].currentBalance.add(poolUserInfo[_poolIndex][msg.sender].pendingBalance) (contracts/poolv1.sol#392)
                - poolUserInfo[_poolIndex][msg.sender].currentPool = poolInfo[_poolIndex].currentRebalance (contracts/poolv1.sol#393)
                - poolUserInfo[_poolIndex][msg.sender].pendingBalance = _amount (contracts/poolv1.sol#394)
                - poolUserInfo[_poolIndex][msg.sender].pendingBalance = poolUserInfo[_poolIndex][msg.sender].pendingBalance.add(_amount) (contracts/poolv1.sol#397)
        - poolUserInfo[_poolIndex][msg.sender].Itokens = poolUserInfo[_poolIndex][msg.sender].Itokens.add(ItokenValue) (contracts/poolv1.sol#577)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
PoolV1.chargePerformancefees(uint256,uint256) (contracts/poolv1.sol#426-449) ignores return value by IERC20(baseStableCoin).transfer(managerAddresses,distribution) (contracts/poolv1.sol#437)
PoolV1.chargePerformancefees(uint256,uint256) (contracts/poolv1.sol#426-449) ignores return value by IERC20(baseStableCoin).transfer(poolInfo[_poolIndex].owner,distribution) (contracts/poolv1.sol#439)
PoolV1.chargePerformancefees(uint256,uint256) (contracts/poolv1.sol#426-449) ignores return value by IERC20(baseStableCoin).transfer(distributor,fees.sub(distribution)) (contracts/poolv1.sol#445)
PoolV1.poolIn(address[],uint256[],uint256) (contracts/poolv1.sol#495-579) ignores return value by IERC20(baseStableCoin).transferFrom(msg.sender,address(this),stableValue) (contracts/poolv1.sol#544)
PoolV1.poolIn(address[],uint256[],uint256) (contracts/poolv1.sol#495-579) ignores return value by IERC20(_tokens[0]).transferFrom(msg.sender,address(this),_values[0]) (contracts/poolv1.sol#546)
PoolV1.withdrawStable(uint256,bool) (contracts/poolv1.sol#646-659) ignores return value by IERC20(baseStableCoin).transfer(distributor,_amount) (contracts/poolv1.sol#653)
PoolV1.withdrawStable(uint256,bool) (contracts/poolv1.sol#646-659) ignores return value by IERC20(baseStableCoin).transfer(msg.sender,_amount) (contracts/poolv1.sol#657)
PoolV1.withdrawPendingAmount(uint256,uint256) (contracts/poolv1.sol#715-724) ignores return value by IERC20(baseStableCoin).transfer(msg.sender,_pendingAmount.sub(_earlyfee)) (contracts/poolv1.sol#721)
PoolV1.chargeEarlyFees(uint256,bool,uint256) (contracts/poolv1.sol#732-760) ignores return value by IERC20(baseStableCoin).transfer(distributor,earlyfees) (contracts/poolv1.sol#743)
PoolV1.chargeEarlyFees(uint256,bool,uint256) (contracts/poolv1.sol#732-760) ignores return value by IERC20(baseStableCoin).transfer(managerAddresses,distribution) (contracts/poolv1.sol#747)
PoolV1.chargeEarlyFees(uint256,bool,uint256) (contracts/poolv1.sol#732-760) ignores return value by IERC20(baseStableCoin).transfer(poolInfo[_poolIndex].owner,distribution) (contracts/poolv1.sol#749)
PoolV1.chargeEarlyFees(uint256,bool,uint256) (contracts/poolv1.sol#732-760) ignores return value by IERC20(baseStableCoin).transfer(distributor,earlyfees.sub(distribution)) (contracts/poolv1.sol#757)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
```

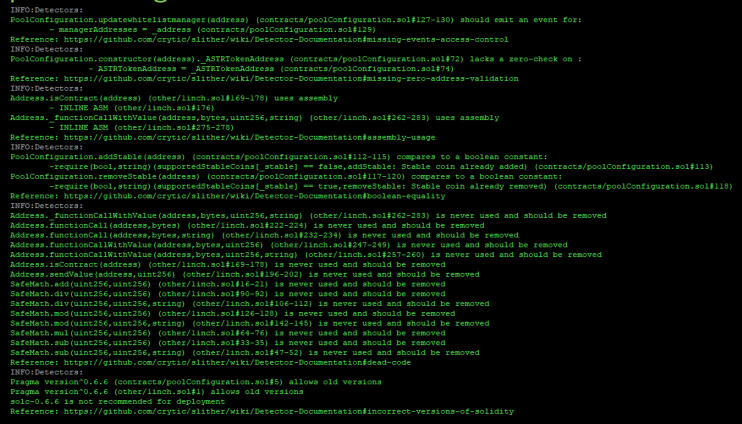


```
INFO:Detectors:
PoolV1.chargePerformancefees(uint256,uint256) (contracts/poolv1.sol#426-449) performs a multiplication on the result of a division:
        -fees = _amount.mul(perFees).div(100) (contracts/poolv1.sol#430)
        -distribution = fees.mul(80).div(100) (contracts/poolv1.sol#431)
PoolV1.withdrawTokens(uint256,uint256) (contracts/poolv1.sol#667-707) performs a multiplication on the result of a division:
        -localWeight = _balance.mul(1000000000000000000).div(totalPoolbalance[_poolIndex]) (contracts/poolv1.sol#672)
        -withdrawBalance = tokenBalance.mul(localWeight).div(1000000000000000000) (contracts/poolv1.sol#685)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
PoolV1.chargeEarlyFees(uint256,bool,uint256) (contracts/poolv1.sol#732-760) uses a dangerous strict equality:
        - poolInfo[_poolIndex].owner == address(this) (contracts/poolv1.sol#746)
PoolV1.chargePerformancefees(uint256,uint256) (contracts/poolv1.sol#426-449) uses a dangerous strict equality:
        - poolInfo[_poolIndex].owner == address(this) (contracts/poolv1.sol#436)
PoolV1.getItokenValue(uint256,uint256,uint256,uint256) (contracts/poolv1.sol#473-487) uses a dangerous strict equality:
        - indexValue == uint256(0) (contracts/poolv1.sol#479)
PoolV1.getPoolValue(uint256) (contracts/poolv1.sol#876-895) uses a dangerous strict equality:
        - _amount == 0 (contracts/poolv1.sol#887)
PoolV1.poolIn(address[],uint256[],uint256) (contracts/poolv1.sol#495-579) uses a dangerous strict equality:
        - poolInfo[_poolIndex].currentRebalance == 0 (contracts/poolv1.sol#568)
PoolV1.rebalance(address[],uint256[],uint256,uint256) (contracts/poolv1.sol#824-870) uses a dangerous strict equality:
        - ethValue == 0 (contracts/poolv1.sol#852)
PoolV1.sellTokensForStable(address[],uint256[]) (contracts/poolv1.sol#1004-1042) uses a dangerous strict equality:
        - _tokens[i] == baseStableCoin (contracts/poolv1.sol#1019)
PoolV1.sellTokensForStable(address[],uint256[]) (contracts/poolv1.sol#1004-1042) uses a dangerous strict equality:
        - _amount == 0 (contracts/poolv1.sol#1029)
PoolV1.swap2(address,uint256,address[],uint256[],uint256,uint256[]) (contracts/poolv1.sol#936-963) uses a dangerous strict equality:
        - _tokenPart == 0 (contracts/poolv1.sol#951)
PoolV1.updatePool(address[],uint256[],uint256,uint256,uint256) (contracts/poolv1.sol#770-811) uses a dangerous strict equality:
        - require(bool,string)(poolInfo[_poolIndex].owner == msg.sender,Only owner can update the punlic pool) (contracts/poolv1.sol#777)
PoolV1.withdrawTokens(uint256,uint256) (contracts/poolv1.sol#667-707) uses a dangerous strict equality:
        - poolInfo[_poolIndex].tokens[i] == baseStableCoin (contracts/poolv1.sol#690)
PoolV1.withdrawTokens(uint256,uint256) (contracts/poolv1.sol#667-707) uses a dangerous strict equality:
        - _amount == 0 (contracts/poolv1.sol#698)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in PoolV1.addNewList() (contracts/poolv1.sol#251-290):
        External calls:
        - (_tokens,_weights,_threshold,_rebalanceTime) = IOracle(IPoolConfiguration(_poolConf).getoracleaddress()).getTokenDetails(_poolIndex) (contracts/poolv1.sol#262)
        - (_name,_symbol) = IOracle(IPoolConfiguration(_poolConf).getoracleaddress()).getiTokenDetails(_poolIndex) (contracts/poolv1.sol#264)
        - _itokenaddr = Iitokendeployer(itokendeployer).createnewitoken(_name,_symbol) (contracts/poolv1.sol#274)
        State variables written after the call(s):
        - poolInfo.push(PoolInfo(_tokens,_weights,_totalWeight,true,_rebalanceTime,0,_threshold,block.timestamp,_itokenaddr,address(this),IOracle(IPoolConfiguration(_poolConf).getoracleaddress()).getindexDescription(_poolIndex))) (cont
cts/poolv1.sol#277-289)
Reentrancy in PoolV1.buytokens(uint256) (contracts/poolv1.sol#357-379):
        External calls:
        - (returnedTokens,returnedAmounts) = swap2(baseStableCoin,ethValue,poolInfo[_poolIndex].tokens,poolInfo[_poolIndex].weights,poolInfo[_poolIndex].totalWeight,buf) (contracts/poolv1.sol#367)
                - IERC20(_token).approve(EXCHANGE_CONTRACT,_value) (contracts/poolv1.sol#945)
                - _amount = IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(_token),IERC20(newTokens[i]),_tokenPart,minReturn,_distribution,0) (contracts/poolv1.sol#960)
        State variables written after the call(s):
        - poolInfo[_poolIndex].currentRebalance = poolInfo[_poolIndex].currentRebalance.add(1) (contracts/poolv1.sol#376)
        - poolPendingbalance[_poolIndex] = 0 (contracts/poolv1.sol#374)
Reentrancy in PoolV1.rebalance(address[],uint256[],uint256,uint256) (contracts/poolv1.sol#824-870):
        External calls:
        - ethValue = sellTokensForStable(poolInfo[_poolIndex].tokens,buf) (contracts/poolv1.sol#841)
                - IERC20(_tokens[i]).approve(EXCHANGE_CONTRACT,_amounts[i]) (contracts/poolv1.sol#1025)
                - _amount = IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(_tokens[i]),IERC20(baseStableCoin),_amounts[i],minReturn,_distribution,0) (contracts/poolv1.sol#1036)
        State variables written after the call(s):
        - buf = buf3 (contracts/poolv1.sol#857)
        - poolInfo[_poolIndex].tokens = newTokens (contracts/poolv1.sol#845)
        - poolInfo[_poolIndex].weights = newWeights (contracts/poolv1.sol#846)
        - poolInfo[_poolIndex].totalWeight = newTotalWeight (contracts/poolv1.sol#847)
        - poolInfo[_poolIndex].currentRebalance = poolInfo[_poolIndex].currentRebalance.add(1) (contracts/poolv1.sol#848)
        - poolInfo[_poolIndex].lastrebalance = block.timestamp (contracts/poolv1.sol#849)
Reentrancy in PoolV1.rebalance(address[],uint256[],uint256,uint256) (contracts/poolv1.sol#824-870):
        External calls:
        - ethValue = sellTokensForStable(poolInfo[_poolIndex].tokens,buf) (contracts/poolv1.sol#841)
                - IERC20(_tokens[i]).approve(EXCHANGE_CONTRACT,_amounts[i]) (contracts/poolv1.sol#1025)
                - _amount = IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(_tokens[i]),IERC20(baseStableCoin),_amounts[i],minReturn,_distribution,0) (contracts/poolv1.sol#1036)
        - (returnedTokens,returnedAmounts) = swap2(baseStableCoin,ethValue,newTokens,newWeights,newTotalWeight,buf) (contracts/poolv1.sol#862)
                - IERC20(_token).approve(EXCHANGE_CONTRACT,_value) (contracts/poolv1.sol#945)
                - _amount = IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(_token),IERC20(newTokens[i]),_tokenPart,minReturn,_distribution,0) (contracts/poolv1.sol#960)
        State variables written after the call(s):
        - (returnedTokens,returnedAmounts) = swap2(baseStableCoin,ethValue,newTokens,newWeights,newTotalWeight,buf) (contracts/poolv1.sol#862)
                - buf = _buf (contracts/poolv1.sol#941)
                - i < newTokens.length (contracts/poolv1.sol#947)
                - buf.push(0) (contracts/poolv1.sol#952)
                - buf.push(_amount) (contracts/poolv1.sol#958)
Reentrancy in PoolV1.updatePool(address[],uint256[],uint256,uint256,uint256) (contracts/poolv1.sol#770-811):
        External calls:
        - (_tokens,_weights,_threshold,_rebalanceTime) = IOracle(IPoolConfiguration(_poolConf).getoracleaddress()).getTokenDetails(_poolIndex) (contracts/poolv1.sol#779)
        - rebalance(newTokens,newWeights,newTotalWeight,_poolIndex) (contracts/poolv1.sol#800)
                - IERC20(_token).approve(EXCHANGE_CONTRACT,_value) (contracts/poolv1.sol#945)
                - IERC20(_tokens[i]).approve(EXCHANGE_CONTRACT,_amounts[i]) (contracts/poolv1.sol#1025)
                - _amount = IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(_token),IERC20(newTokens[i]),_tokenPart,minReturn,_distribution,0) (contracts/poolv1.sol#960)
                - _amount = IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(_tokens[i]),IERC20(baseStableCoin),_amounts[i],minReturn,_distribution,0) (contracts/poolv1.sol#1036)
        State variables written after the call(s):
        - rebalance(newTokens,newWeights,newTotalWeight,_poolIndex) (contracts/poolv1.sol#800)
                - poolInfo[_poolIndex].tokens = newTokens (contracts/poolv1.sol#845)
                - poolInfo[_poolIndex].weights = newWeights (contracts/poolv1.sol#846)
                - poolInfo[_poolIndex].totalWeight = newTotalWeight (contracts/poolv1.sol#847)
                - poolInfo[_poolIndex].currentRebalance = poolInfo[_poolIndex].currentRebalance.add(1) (contracts/poolv1.sol#848)
                - poolInfo[_poolIndex].lastrebalance = block.timestamp (contracts/poolv1.sol#849)
        - poolInfo[_poolIndex].threshold = _threshold (contracts/poolv1.sol#803)
        - poolInfo[_poolIndex].rebaltime = _rebalanceTime (contracts/poolv1.sol#804)
Reentrancy in PoolV1.updatePool(address[],uint256[],uint256,uint256,uint256) (contracts/poolv1.sol#770-811):
        External calls:
        - (_tokens,_weights,_threshold,_rebalanceTime) = IOracle(IPoolConfiguration(_poolConf).getoracleaddress()).getTokenDetails(_poolIndex) (contracts/poolv1.sol#779)
        - rebalance(newTokens,newWeights,newTotalWeight,_poolIndex) (contracts/poolv1.sol#800)
                - IERC20(_token).approve(EXCHANGE_CONTRACT,_value) (contracts/poolv1.sol#945)
                - IERC20(_tokens[i]).approve(EXCHANGE_CONTRACT,_amounts[i]) (contracts/poolv1.sol#1025)
                - _amount = IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(_token),IERC20(newTokens[i]),_tokenPart,minReturn,_distribution,0) (contracts/poolv1.sol#960)
                - _amount = IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(_tokens[i]),IERC20(baseStableCoin),_amounts[i],minReturn,_distribution,0) (contracts/poolv1.sol#1036)
        - buytokens(_poolIndex) (contracts/poolv1.sol#808)
                - IERC20(_token).approve(EXCHANGE_CONTRACT,_value) (contracts/poolv1.sol#945)
                - _amount = IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(_token),IERC20(newTokens[i]),_tokenPart,minReturn,_distribution,0) (contracts/poolv1.sol#960)
        State variables written after the call(s):
        - buytokens(_poolIndex) (contracts/poolv1.sol#808)
                - buf = _buf (contracts/poolv1.sol#941)
                - buf = buf3 (contracts/poolv1.sol#365)
                - i < newTokens.length (contracts/poolv1.sol#947)
                - buf.push(0) (contracts/poolv1.sol#952)
                - buf.push(_amount) (contracts/poolv1.sol#958)
        - buytokens(_poolIndex) (contracts/poolv1.sol#808)
                - poolInfo[_poolIndex].currentRebalance = poolInfo[_poolIndex].currentRebalance.add(1) (contracts/poolv1.sol#376)
        - buytokens(_poolIndex) (contracts/poolv1.sol#808)
                - tokenBalances[_poolIndex][returnedTokens[i]] += returnedAmounts[i] (contracts/poolv1.sol#370)
        - buytokens(_poolIndex) (contracts/poolv1.sol#808)
                - totalPoolbalance[_poolIndex] = totalPoolbalance[_poolIndex].add(ethValue) (contracts/poolv1.sol#373)
```



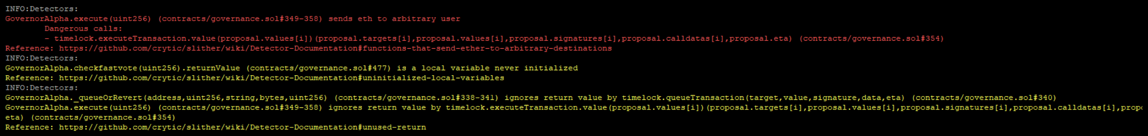
```
Reentrancy in PoolV1.withdraw(uint256,bool,bool,uint256) (contracts/poolv1.sol#588-644):
        External calls:
        - _totalAmount = withdrawTokens(_poolIndex,_balance) (contracts/poolv1.sol#606)
                - IERC20(poolInfo[_poolIndex].tokens[i]).approve(EXCHANGE_CONTRACT,withdrawBalance) (contracts/poolv1.sol#695)
                - IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(poolInfo[_poolIndex].tokens[i]),IERC20(baseStableCoin),withdrawBalance,_amount,_distribution,0) (contracts/poolv1.sol#704)
        - fees = chargePerformancefees(_totalAmount.sub(_balance),_poolIndex) (contracts/poolv1.sol#613)
                - IERC20(baseStableCoin).transfer(managerAddresses,distribution) (contracts/poolv1.sol#437)
                - IERC20(baseStableCoin).transfer(poolInfo[_poolIndex].owner,distribution) (contracts/poolv1.sol#439)
                - IERC20(baseStableCoin).transfer(distributor,fees.sub(distribution)) (contracts/poolv1.sol#445)
        - withdrawStable(_totalAmount.sub(fees).sub(earlyfees),stakePremium) (contracts/poolv1.sol#617)
                - IERC20(baseStableCoin).transfer(distributor,_amount) (contracts/poolv1.sol#653)
                - IERC20(baseStableCoin).transfer(msg.sender,_amount) (contracts/poolv1.sol#657)
        - pendingEarlyfees = withdrawPendingAmount(_poolIndex,_pendingAmount) (contracts/poolv1.sol#619)
                - IERC20(baseStableCoin).transfer(msg.sender,_pendingAmount.sub(_earlyfee)) (contracts/poolv1.sol#721)
        - chargeEarlyFees(earlyfees.add(pendingEarlyfees),stakeEarlyFees,_poolIndex) (contracts/poolv1.sol#621)
                - IERC20(baseStableCoin).transfer(distributor,earlyfees) (contracts/poolv1.sol#743)
                - IERC20(baseStableCoin).transfer(managerAddresses,distribution) (contracts/poolv1.sol#747)
                - IERC20(baseStableCoin).transfer(poolInfo[_poolIndex].owner,distribution) (contracts/poolv1.sol#749)
                - IERC20(baseStableCoin).transfer(distributor,earlyfees.sub(distribution)) (contracts/poolv1.sol#757)
        - withdrawStable(_totalAmount.sub(earlyfees_scope_0),stakePremium) (contracts/poolv1.sol#628)
                - IERC20(baseStableCoin).transfer(distributor,_amount) (contracts/poolv1.sol#653)
                - IERC20(baseStableCoin).transfer(msg.sender,_amount) (contracts/poolv1.sol#657)
        - pendingEarlyfees_scope_1 = withdrawPendingAmount(_poolIndex,_pendingAmount) (contracts/poolv1.sol#630)
                - IERC20(baseStableCoin).transfer(msg.sender,_pendingAmount.sub(_earlyfee)) (contracts/poolv1.sol#721)
        - chargeEarlyFees(earlyfees_scope_0.add(pendingEarlyfees_scope_1),stakeEarlyFees,_poolIndex) (contracts/poolv1.sol#632)
                - IERC20(baseStableCoin).transfer(distributor,earlyfees) (contracts/poolv1.sol#743)
                - IERC20(baseStableCoin).transfer(managerAddresses,distribution) (contracts/poolv1.sol#747)
                - IERC20(baseStableCoin).transfer(poolInfo[_poolIndex].owner,distribution) (contracts/poolv1.sol#749)
                - IERC20(baseStableCoin).transfer(distributor,earlyfees.sub(distribution)) (contracts/poolv1.sol#757)
        State variables written after the call(s):
        - poolUserInfo[_poolIndex][msg.sender].Itokens = poolUserInfo[_poolIndex][msg.sender].Itokens.sub(withdrawAmount) (contracts/poolv1.sol#636)
Reentrancy in PoolV1.withdraw(uint256,bool,bool,uint256) (contracts/poolv1.sol#588-644):
        External calls:
        - _totalAmount = withdrawTokens(_poolIndex,_balance) (contracts/poolv1.sol#606)
                - IERC20(poolInfo[_poolIndex].tokens[i]).approve(EXCHANGE_CONTRACT,withdrawBalance) (contracts/poolv1.sol#695)
                - IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(poolInfo[_poolIndex].tokens[i]),IERC20(baseStableCoin),withdrawBalance,_amount,_distribution,0) (contracts/poolv1.sol#704)
        - fees = chargePerformancefees(_totalAmount.sub(_balance),_poolIndex) (contracts/poolv1.sol#613)
                - IERC20(baseStableCoin).transfer(managerAddresses,distribution) (contracts/poolv1.sol#437)
                - IERC20(baseStableCoin).transfer(poolInfo[_poolIndex].owner,distribution) (contracts/poolv1.sol#439)
                - IERC20(baseStableCoin).transfer(distributor,fees.sub(distribution)) (contracts/poolv1.sol#445)
        - withdrawStable(_totalAmount.sub(fees).sub(earlyfees),stakePremium) (contracts/poolv1.sol#617)
                - IERC20(baseStableCoin).transfer(distributor,_amount) (contracts/poolv1.sol#653)
                - IERC20(baseStableCoin).transfer(msg.sender,_amount) (contracts/poolv1.sol#657)
        - pendingEarlyfees = withdrawPendingAmount(_poolIndex,_pendingAmount) (contracts/poolv1.sol#619)
                - IERC20(baseStableCoin).transfer(msg.sender,_pendingAmount.sub(_earlyfee)) (contracts/poolv1.sol#721)
        - chargeEarlyFees(earlyfees.add(pendingEarlyfees),stakeEarlyFees,_poolIndex) (contracts/poolv1.sol#621)
                - IERC20(baseStableCoin).transfer(distributor,earlyfees) (contracts/poolv1.sol#743)
                - IERC20(baseStableCoin).transfer(managerAddresses,distribution) (contracts/poolv1.sol#747)
                - IERC20(baseStableCoin).transfer(poolInfo[_poolIndex].owner,distribution) (contracts/poolv1.sol#749)
                - IERC20(baseStableCoin).transfer(distributor,earlyfees.sub(distribution)) (contracts/poolv1.sol#757)
        - withdrawStable(_totalAmount.sub(earlyfees_scope_0),stakePremium) (contracts/poolv1.sol#628)
                - IERC20(baseStableCoin).transfer(distributor,_amount) (contracts/poolv1.sol#653)
                - IERC20(baseStableCoin).transfer(msg.sender,_amount) (contracts/poolv1.sol#657)
        - pendingEarlyfees_scope_1 = withdrawPendingAmount(_poolIndex,_pendingAmount) (contracts/poolv1.sol#630)
                - IERC20(baseStableCoin).transfer(msg.sender,_pendingAmount.sub(_earlyfee)) (contracts/poolv1.sol#721)
        - chargeEarlyFees(earlyfees_scope_0.add(pendingEarlyfees_scope_1),stakeEarlyFees,_poolIndex) (contracts/poolv1.sol#632)
                - IERC20(baseStableCoin).transfer(distributor,earlyfees) (contracts/poolv1.sol#743)
                - IERC20(baseStableCoin).transfer(managerAddresses,distribution) (contracts/poolv1.sol#747)
                - IERC20(baseStableCoin).transfer(poolInfo[_poolIndex].owner,distribution) (contracts/poolv1.sol#749)
                - IERC20(baseStableCoin).transfer(distributor,earlyfees.sub(distribution)) (contracts/poolv1.sol#757)
        - Iitoken(poolInfo[_poolIndex].itokenaddr).burn(msg.sender,withdrawAmount) (contracts/poolv1.sol#637)

        State variables written after the call(s):
        - poolUserInfo[_poolIndex][msg.sender].pendingBalance = poolUserInfo[_poolIndex][msg.sender].pendingBalance.sub(_pendingAmount) (contracts/poolv1.sol#640)
        - poolUserInfo[_poolIndex][msg.sender].currentBalance = poolUserInfo[_poolIndex][msg.sender].currentBalance.sub(_balance) (contracts/poolv1.sol#642)
        - totalPoolbalance[_poolIndex] = totalPoolbalance[_poolIndex].sub(_balance) (contracts/poolv1.sol#641)
Reentrancy in PoolV1.withdrawTokens(uint256,uint256) (contracts/poolv1.sol#667-707):
        External calls:
        - IERC20(poolInfo[_poolIndex].tokens[i]).approve(EXCHANGE_CONTRACT,withdrawBalance) (contracts/poolv1.sol#695)
        State variables written after the call(s):
        - tokenBalances[_poolIndex][poolInfo[_poolIndex].tokens[i]] = tokenBalance.sub(withdrawBalance) (contracts/poolv1.sol#702)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
PoolV1.rebalance(address[],uint256[],uint256,uint256).buf3 (contracts/poolv1.sol#856) is a local variable never initialized
PoolV1.withdrawTokens(uint256,uint256).localWeight (contracts/poolv1.sol#668) is a local variable never initialized
PoolV1.poolIn(address[],uint256[],uint256).returnedTokens (contracts/poolv1.sol#511) is a local variable never initialized
PoolV1.buytokens(uint256).buf3 (contracts/poolv1.sol#364) is a local variable never initialized
PoolV1.withdraw(uint256,bool,bool,uint256).earlyfees_scope_0 (contracts/poolv1.sol#625) is a local variable never initialized
PoolV1.poolIn(address[],uint256[],uint256).returnedAmounts (contracts/poolv1.sol#512) is a local variable never initialized
PoolV1.withdrawPendingAmount(uint256,uint256)._earlyfee (contracts/poolv1.sol#716) is a local variable never initialized
PoolV1.rebalance(address[],uint256[],uint256,uint256).ethValue (contracts/poolv1.sol#829) is a local variable never initialized
PoolV1.withdraw(uint256,bool,bool,uint256)._balance (contracts/poolv1.sol#595) is a local variable never initialized
PoolV1.buyAstraToken(uint256)._amount (contracts/poolv1.sol#298) is a local variable never initialized
PoolV1.withdraw(uint256,bool,bool,uint256).earlyfees (contracts/poolv1.sol#611) is a local variable never initialized
PoolV1.rebalance(address[],uint256[],uint256,uint256).buf2 (contracts/poolv1.sol#827) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
PoolV1.buyAstraToken(uint256) (contracts/poolv1.sol#297-307) ignores return value by IERC20(baseStableCoin).approve(EXCHANGE_CONTRACT,_amount) (contracts/poolv1.sol#300)
PoolV1.buyAstraToken(uint256) (contracts/poolv1.sol#297-307) ignores return value by IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(baseStableCoin),IERC20(ASTRTokenAddress),_Amount,minReturn,_distribution,0) (contracts/poolv1.sol#305)
PoolV1.stakeAstra(uint256,bool) (contracts/poolv1.sol#314-319) ignores return value by IERC20(ASTRTokenAddress).approve(address(_poolConf),_amount) (contracts/poolv1.sol#316)
PoolV1.poolIn(address[],uint256[],uint256) (contracts/poolv1.sol#495-579) ignores return value by Iitoken(poolInfo[_poolIndex].itokenaddr).mint(msg.sender,ItokenValue) (contracts/poolv1.sol#578)
PoolV1.withdraw(uint256,bool,bool,uint256) (contracts/poolv1.sol#588-644) ignores return value by Iitoken(poolInfo[_poolIndex].itokenaddr).burn(msg.sender,withdrawAmount) (contracts/poolv1.sol#637)
PoolV1.withdrawTokens(uint256,uint256) (contracts/poolv1.sol#667-707) ignores return value by IERC20(poolInfo[_poolIndex].tokens[i]).approve(EXCHANGE_CONTRACT,withdrawBalance) (contracts/poolv1.sol#695)
PoolV1.withdrawTokens(uint256,uint256) (contracts/poolv1.sol#667-707) ignores return value by IOneSplit(EXCHANGE_CONTRACT).swap(IERC20(poolInfo[_poolIndex].tokens[i]),IERC20(baseStableCoin),withdrawBalance,_amount,_distribution,0) (con
acts/poolv1.sol#704)
PoolV1.swap(address,uint256,address[],uint256[],uint256) (contracts/poolv1.sol#901-930) ignores return value by IERC20(_tokens[i]).approve(EXCHANGE_CONTRACT,_tokenPart) (contracts/poolv1.sol#923)
PoolV1.swap2(address,uint256,address[],uint256[],uint256,uint256[]) (contracts/poolv1.sol#936-963) ignores return value by IERC20(_token).approve(EXCHANGE_CONTRACT,_value) (contracts/poolv1.sol#945)
PoolV1.sellTokensForEther(address[],uint256[]) (contracts/poolv1.sol#968-999) ignores return value by IERC20(_tokens[i]).approve(EXCHANGE_CONTRACT,_amounts[i]) (contracts/poolv1.sol#983)
PoolV1.sellTokensForStable(address[],uint256[]) (contracts/poolv1.sol#1004-1042) ignores return value by IERC20(_tokens[i]).approve(EXCHANGE_CONTRACT,_amounts[i]) (contracts/poolv1.sol#1025)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

poolConfiguration.sol



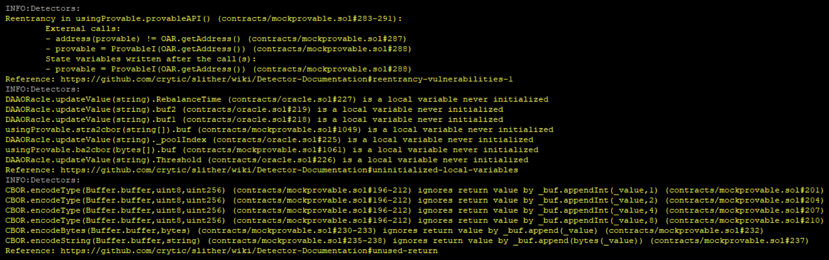
```
INFO:Detectors:
PoolConfiguration.updatewhitelistmanager(address) (contracts/poolConfiguration.sol#127-130) should emit an event for:
        - managerAddresses = _address (contracts/poolConfiguration.sol#129)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control
INFO:Detectors:
PoolConfiguration.constructor(address)._ASTRTokenAddress (contracts/poolConfiguration.sol#72) lacks a zero-check on :
                - ASTRTokenAddress = _ASTRTokenAddress (contracts/poolConfiguration.sol#74)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Address.isContract(address) (other/linch.sol#169-178) uses assembly
        - INLINE ASM (other/linch.sol#176)
Address._functionCallWithValue(address,bytes,uint256,string) (other/linch.sol#262-283) uses assembly
        - INLINE ASM (other/linch.sol#275-278)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
PoolConfiguration.addStable(address) (contracts/poolConfiguration.sol#112-115) compares to a boolean constant:
        -require(bool,string)(supportedStableCoins[_stable] == false,addStable: Stable coin already added) (contracts/poolConfiguration.sol#113)
PoolConfiguration.removeStable(address) (contracts/poolConfiguration.sol#117-120) compares to a boolean constant:
        -require(bool,string)(supportedStableCoins[_stable] == true,removeStable: Stable coin already removed) (contracts/poolConfiguration.sol#118)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Address._functionCallWithValue(address,bytes,uint256,string) (other/linch.sol#262-283) is never used and should be removed
Address.functionCall(address,bytes) (other/linch.sol#222-224) is never used and should be removed
Address.functionCall(address,bytes,string) (other/linch.sol#232-234) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (other/linch.sol#247-249) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (other/linch.sol#257-260) is never used and should be removed
Address.isContract(address) (other/linch.sol#169-178) is never used and should be removed
Address.sendValue(address,uint256) (other/linch.sol#196-202) is never used and should be removed
SafeMath.add(uint256,uint256) (other/linch.sol#16-21) is never used and should be removed
SafeMath.div(uint256,uint256) (other/linch.sol#90-92) is never used and should be removed
SafeMath.div(uint256,uint256,string) (other/linch.sol#106-112) is never used and should be removed
SafeMath.mod(uint256,uint256) (other/linch.sol#126-128) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (other/linch.sol#142-145) is never used and should be removed
SafeMath.mul(uint256,uint256) (other/linch.sol#64-76) is never used and should be removed
SafeMath.sub(uint256,uint256) (other/linch.sol#33-35) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (other/linch.sol#47-52) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.6.6 (contracts/poolConfiguration.sol#5) allows old versions
Pragma version^0.6.6 (other/linch.sol#1) allows old versions
solc-0.6.6 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

governance.sol



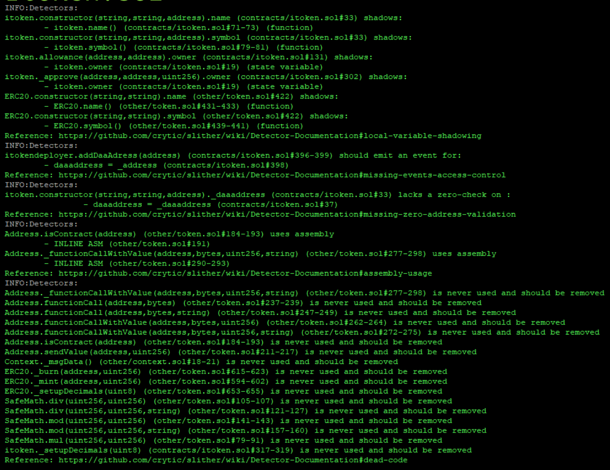
```
INFO:Detectors:
GovernorAlpha.execute(uint256) (contracts/governance.sol#349-358) sends eth to arbitrary user
        Dangerous calls:
        - timelock.executeTransaction.value(proposal.values[i])(proposal.targets[i],proposal.values[i],proposal.signatures[i],proposal.calldatas[i],proposal.eta) (contracts/governance.sol#354)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
GovernorAlpha.checkfastvote(uint256).returnValue (contracts/governance.sol#477) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
GovernorAlpha._queueOrRevert(address,uint256,string,bytes,uint256) (contracts/governance.sol#338-341) ignores return value by timelock.queueTransaction(target,value,signature,data,eta) (contracts/governance.sol#340)
GovernorAlpha.execute(uint256) (contracts/governance.sol#349-358) ignores return value by timelock.executeTransaction.value(proposal.values[i])(proposal.targets[i],proposal.values[i],proposal.signatures[i],proposal.calldatas[i],propo
eta) (contracts/governance.sol#354)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

oracle.sol



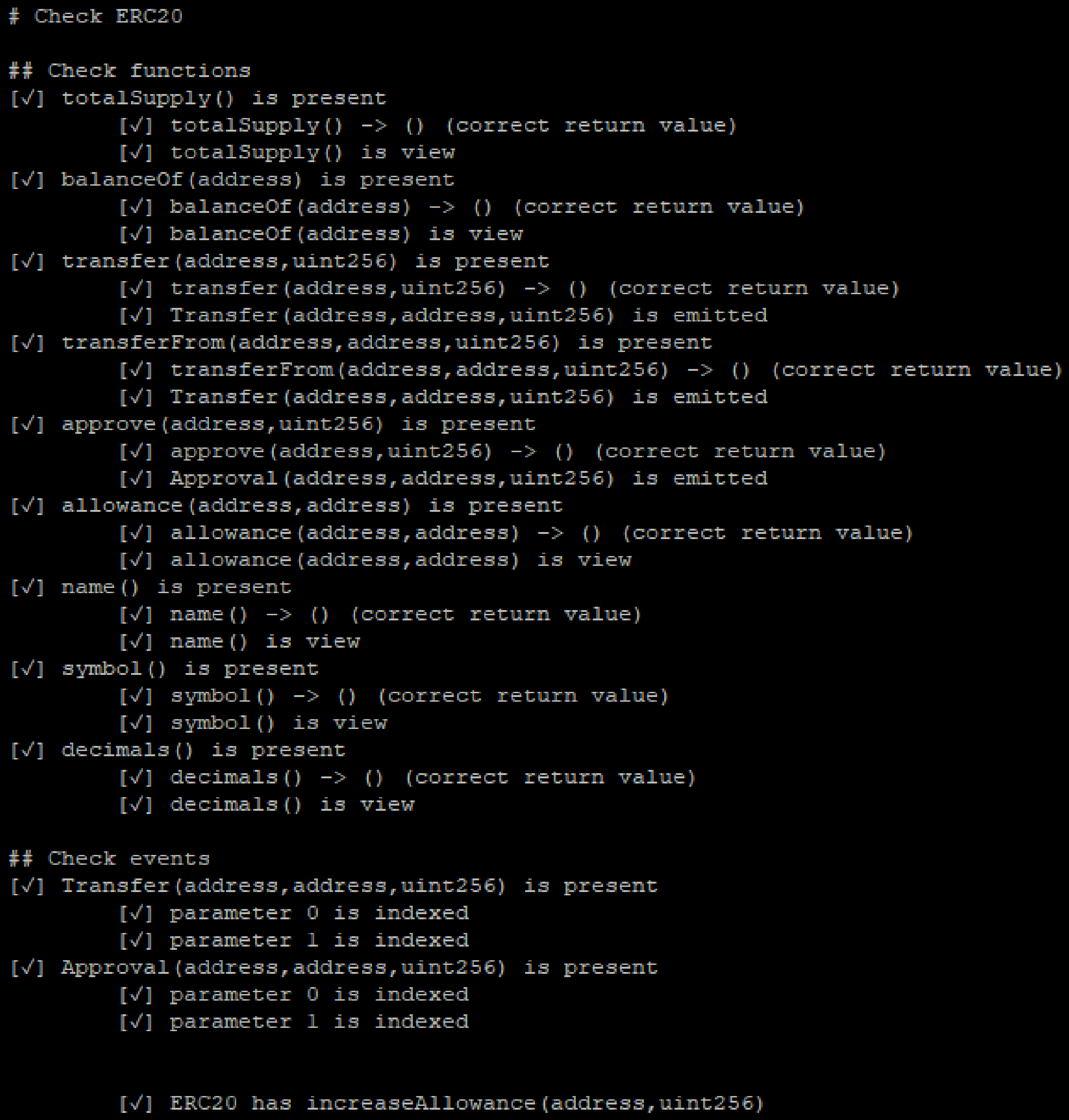
```
INFO:Detectors:
Reentrancy in usingProvable.provableAPI() (contracts/mockprovable.sol#283-291):
        External calls:
        - address(provable) != OAR.getAddress() (contracts/mockprovable.sol#287)
        - provable = ProvableI(OAR.getAddress()) (contracts/mockprovable.sol#288)
        State variables written after the call(s):
        - provable = ProvableI(OAR.getAddress()) (contracts/mockprovable.sol#288)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
DAAORacle.updateValue(string).RebalanceTime (contracts/oracle.sol#227) is a local variable never initialized
DAAORacle.updateValue(string).buf2 (contracts/oracle.sol#219) is a local variable never initialized
DAAORacle.updateValue(string).buf1 (contracts/oracle.sol#218) is a local variable never initialized
usingProvable.stra2cbor(string[]).buf (contracts/mockprovable.sol#1049) is a local variable never initialized
DAAORacle.updateValue(string)._poolIndex (contracts/oracle.sol#225) is a local variable never initialized
usingProvable.ba2cbor(bytes[]).buf (contracts/mockprovable.sol#1061) is a local variable never initialized
DAAORacle.updateValue(string).Threshold (contracts/oracle.sol#226) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
CBOR.encodeType(Buffer.buffer,uint8,uint256) (contracts/mockprovable.sol#196-212) ignores return value by _buf.appendInt(_value,1) (contracts/mockprovable.sol#201)
CBOR.encodeType(Buffer.buffer,uint8,uint256) (contracts/mockprovable.sol#196-212) ignores return value by _buf.appendInt(_value,2) (contracts/mockprovable.sol#204)
CBOR.encodeType(Buffer.buffer,uint8,uint256) (contracts/mockprovable.sol#196-212) ignores return value by _buf.appendInt(_value,4) (contracts/mockprovable.sol#207)
CBOR.encodeType(Buffer.buffer,uint8,uint256) (contracts/mockprovable.sol#196-212) ignores return value by _buf.appendInt(_value,8) (contracts/mockprovable.sol#210)
CBOR.encodeBytes(Buffer.buffer,bytes) (contracts/mockprovable.sol#230-233) ignores return value by _buf.append(_value) (contracts/mockprovable.sol#232)
CBOR.encodeString(Buffer.buffer,string) (contracts/mockprovable.sol#235-238) ignores return value by _buf.append(bytes(_value)) (contracts/mockprovable.sol#237)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

itoken.sol



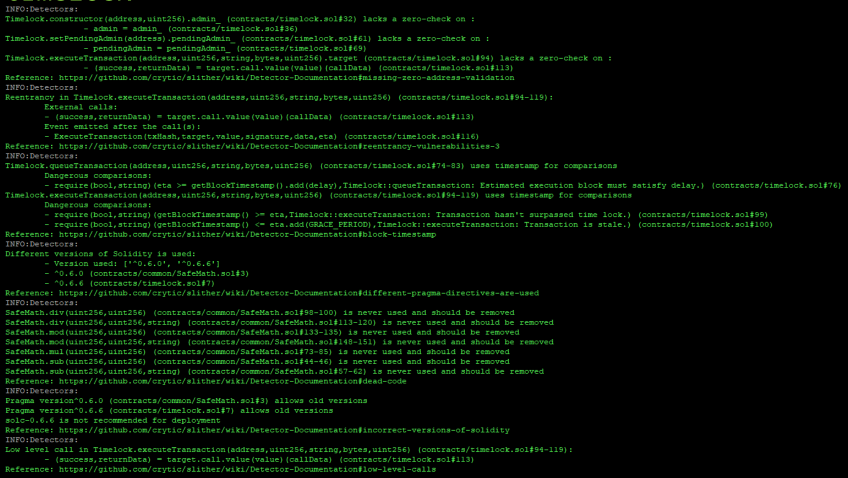
```
INFO:Detectors:
itoken.constructor(string,string,address).name (contracts/itoken.sol#33) shadows:
        - itoken.name() (contracts/itoken.sol#71-73) (function)
itoken.constructor(string,string,address).symbol (contracts/itoken.sol#33) shadows:
        - itoken.symbol() (contracts/itoken.sol#79-81) (function)
itoken.allowance(address,address).owner (contracts/itoken.sol#131) shadows:
        - itoken.owner (contracts/itoken.sol#19) (state variable)
itoken._approve(address,address,uint256).owner (contracts/itoken.sol#302) shadows:
        - itoken.owner (contracts/itoken.sol#19) (state variable)
ERC20.constructor(string,string).name (other/token.sol#422) shadows:
        - ERC20.name() (other/token.sol#431-433) (function)
ERC20.constructor(string,string).symbol (other/token.sol#422) shadows:
        - ERC20.symbol() (other/token.sol#439-441) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
itokendeployer.addDaaAdress(address) (contracts/itoken.sol#396-399) should emit an event for:
        - daaaddress = _address (contracts/itoken.sol#398)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control
INFO:Detectors:
itoken.constructor(string,string,address)._daaaddress (contracts/itoken.sol#33) lacks a zero-check on :
                - daaaddress = _daaaddress (contracts/itoken.sol#37)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Address.isContract(address) (other/token.sol#184-193) uses assembly
        - INLINE ASM (other/token.sol#191)
Address._functionCallWithValue(address,bytes,uint256,string) (other/token.sol#277-298) uses assembly
        - INLINE ASM (other/token.sol#290-293)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address._functionCallWithValue(address,bytes,uint256,string) (other/token.sol#277-298) is never used and should be removed
Address.functionCall(address,bytes) (other/token.sol#237-239) is never used and should be removed
Address.functionCall(address,bytes,string) (other/token.sol#247-249) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (other/token.sol#262-264) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (other/token.sol#272-275) is never used and should be removed
Address.isContract(address) (other/token.sol#184-193) is never used and should be removed
Address.sendValue(address,uint256) (other/token.sol#211-217) is never used and should be removed
Context._msgData() (other/context.sol#18-21) is never used and should be removed
ERC20._burn(address,uint256) (other/token.sol#615-623) is never used and should be removed
ERC20._mint(address,uint256) (other/token.sol#594-602) is never used and should be removed
ERC20._setupDecimals(uint8) (other/token.sol#653-655) is never used and should be removed
SafeMath.div(uint256,uint256) (other/token.sol#105-107) is never used and should be removed
SafeMath.div(uint256,uint256,string) (other/token.sol#121-127) is never used and should be removed
SafeMath.mod(uint256,uint256) (other/token.sol#141-143) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (other/token.sol#157-160) is never used and should be removed
SafeMath.mul(uint256,uint256) (other/token.sol#79-91) is never used and should be removed
itoken._setupDecimals(uint8) (contracts/itoken.sol#317-319) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

ERC CONFORMAL CHECKER FOR itoken.sol passed:



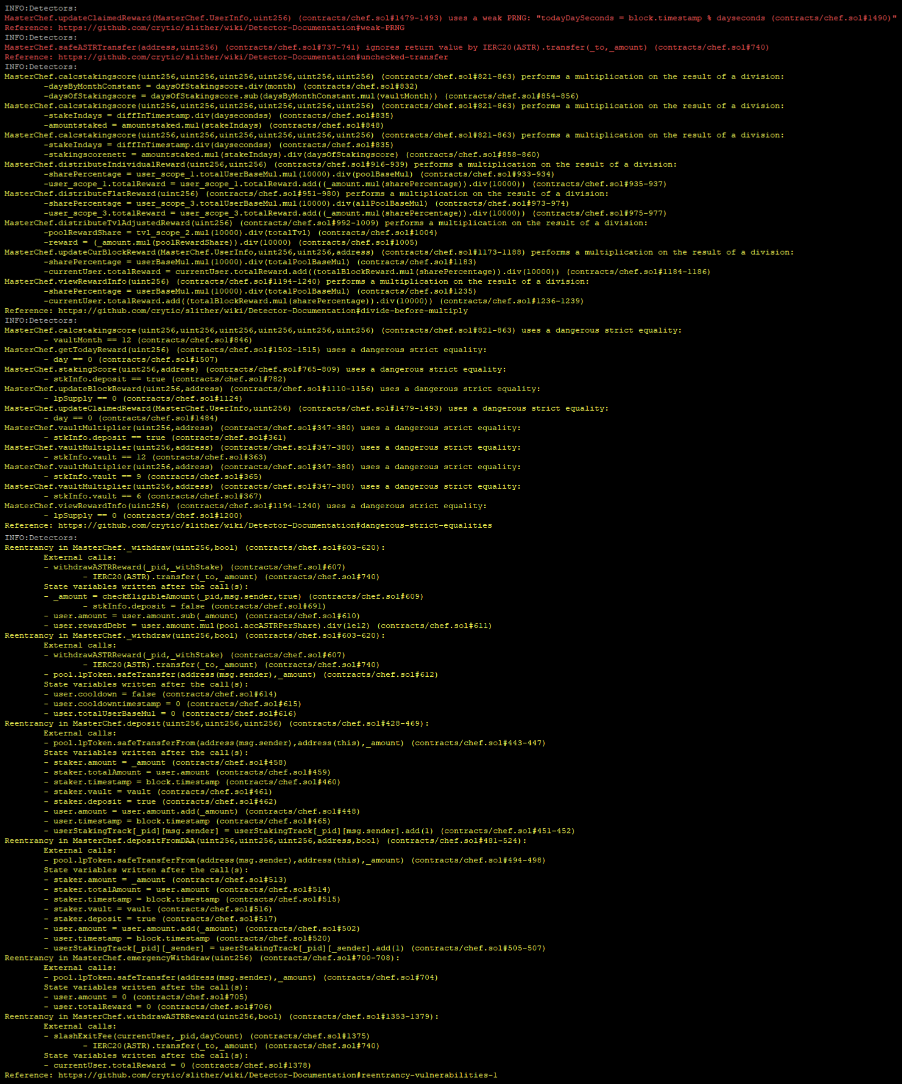
```
# Check ERC20

## Check functions
[✓] totalSupply() is present
        [✓] totalSupply() -> () (correct return value)
        [✓] totalSupply() is view
[✓] balanceOf(address) is present
        [✓] balanceOf(address) -> () (correct return value)
        [✓] balanceOf(address) is view
[✓] transfer(address,uint256) is present
        [✓] transfer(address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] transferFrom(address,address,uint256) is present
        [✓] transferFrom(address,address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] approve(address,uint256) is present
        [✓] approve(address,uint256) -> () (correct return value)
        [✓] Approval(address,address,uint256) is emitted
[✓] allowance(address,address) is present
        [✓] allowance(address,address) -> () (correct return value)
        [✓] allowance(address,address) is view
[✓] name() is present
        [✓] name() -> () (correct return value)
        [✓] name() is view
[✓] symbol() is present
        [✓] symbol() -> () (correct return value)
        [✓] symbol() is view
[✓] decimals() is present
        [✓] decimals() -> () (correct return value)
        [✓] decimals() is view

## Check events
[✓] Transfer(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed
[✓] Approval(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed


        [✓] ERC20 has increaseAllowance(address,uint256)
```

timelock

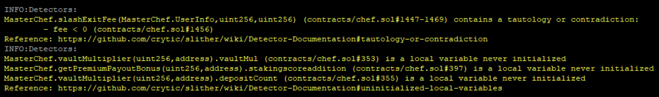


```
INFO:Detectors:
Timelock.constructor(address,uint256).admin_ (contracts/timelock.sol#32) lacks a zero-check on :
                - admin = admin_ (contracts/timelock.sol#36)
Timelock.setPendingAdmin(address).pendingAdmin_ (contracts/timelock.sol#61) lacks a zero-check on :
                - pendingAdmin = pendingAdmin_ (contracts/timelock.sol#69)
Timelock.executeTransaction(address,uint256,string,bytes,uint256).target (contracts/timelock.sol#94) lacks a zero-check on :
                - (success,returnData) = target.call.value(value)(callData) (contracts/timelock.sol#113)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in Timelock.executeTransaction(address,uint256,string,bytes,uint256) (contracts/timelock.sol#94-119):
        External calls:
        - (success,returnData) = target.call.value(value)(callData) (contracts/timelock.sol#113)
        Event emitted after the call(s):
        - ExecuteTransaction(txHash,target,value,signature,data,eta) (contracts/timelock.sol#116)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Timelock.queueTransaction(address,uint256,string,bytes,uint256) (contracts/timelock.sol#74-83) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(eta >= getBlockTimestamp().add(delay),Timelock::queueTransaction: Estimated execution block must satisfy delay.) (contracts/timelock.sol#76)
Timelock.executeTransaction(address,uint256,string,bytes,uint256) (contracts/timelock.sol#94-119) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(getBlockTimestamp() >= eta,Timelock::executeTransaction: Transaction hasn't surpassed time lock.) (contracts/timelock.sol#99)
        - require(bool,string)(getBlockTimestamp() <= eta.add(GRACE_PERIOD),Timelock::executeTransaction: Transaction is stale.) (contracts/timelock.sol#100)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Different versions of Solidity is used:
        - Version used: ['^0.6.0', '^0.6.6']
        - ^0.6.0 (contracts/common/SafeMath.sol#3)
        - ^0.6.6 (contracts/timelock.sol#7)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
SafeMath.div(uint256,uint256) (contracts/common/SafeMath.sol#98-100) is never used and should be removed
SafeMath.div(uint256,uint256,string) (contracts/common/SafeMath.sol#113-120) is never used and should be removed
SafeMath.mod(uint256,uint256) (contracts/common/SafeMath.sol#133-135) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (contracts/common/SafeMath.sol#148-151) is never used and should be removed
SafeMath.mul(uint256,uint256) (contracts/common/SafeMath.sol#73-85) is never used and should be removed
SafeMath.sub(uint256,uint256) (contracts/common/SafeMath.sol#44-46) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (contracts/common/SafeMath.sol#57-62) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.6.0 (contracts/common/SafeMath.sol#3) allows old versions
Pragma version^0.6.6 (contracts/timelock.sol#7) allows old versions
solc-0.6.6 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Timelock.executeTransaction(address,uint256,string,bytes,uint256) (contracts/timelock.sol#94-119):
        - (success,returnData) = target.call.value(value)(callData) (contracts/timelock.sol#113)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

chef.sol

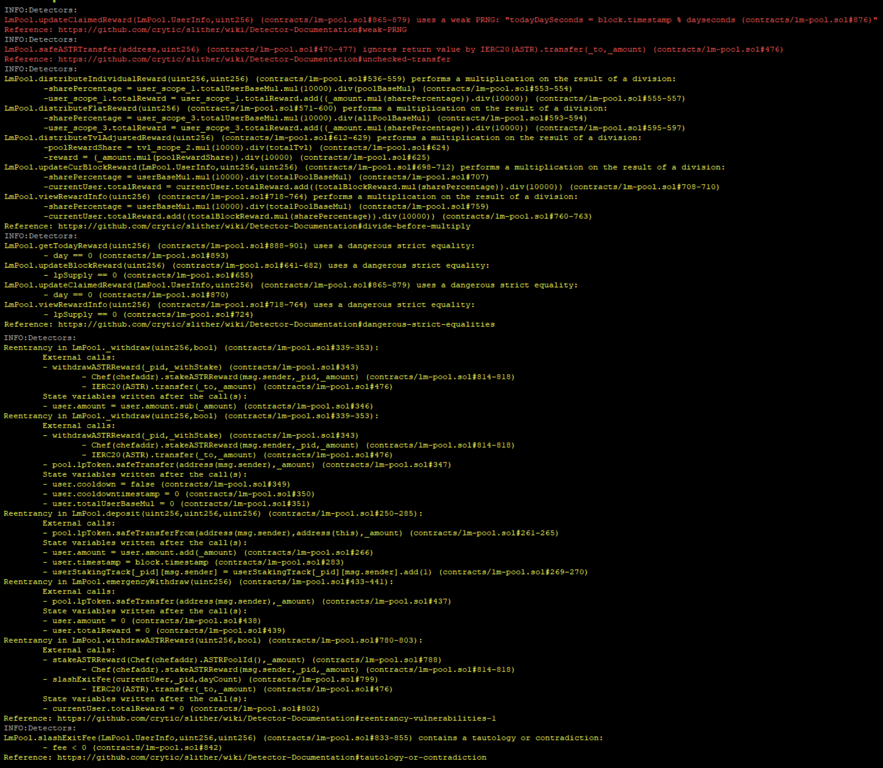


```
INFO:Detectors:
MasterChef.updateClaimedReward(MasterChef.UserInfo,uint256) (contracts/chef.sol#1479-1493) uses a weak PRNG: "todayDaySeconds = block.timestamp % dayseconds (contracts/chef.sol#1490)"
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#weak-PRNG
INFO:Detectors:
MasterChef.safeASTRTransfer(address,uint256) (contracts/chef.sol#737-741) ignores return value by IERC20(ASTR).transfer(_to,_amount) (contracts/chef.sol#740)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
MasterChef.calcstakingscore(uint256,uint256,uint256,uint256,uint256,uint256) (contracts/chef.sol#821-863) performs a multiplication on the result of a division:
        -daysByMonthConstant = daysOfStakingscore.div(month) (contracts/chef.sol#832)
        -daysOfStakingscore = daysOfStakingscore.sub(daysByMonthConstant.mul(vaultMonth)) (contracts/chef.sol#854-856)
MasterChef.calcstakingscore(uint256,uint256,uint256,uint256,uint256,uint256) (contracts/chef.sol#821-863) performs a multiplication on the result of a division:
        -stakeIndays = diffInTimestamp.div(daysecondss) (contracts/chef.sol#835)
        -amountstaked = amountstaked.mul(stakeIndays) (contracts/chef.sol#848)
MasterChef.calcstakingscore(uint256,uint256,uint256,uint256,uint256,uint256) (contracts/chef.sol#821-863) performs a multiplication on the result of a division:
        -stakeIndays = diffInTimestamp.div(daysecondss) (contracts/chef.sol#835)
        -stakingscorenett = amountstaked.mul(stakeIndays).div(daysOfStakingscore) (contracts/chef.sol#858-860)
MasterChef.distributeIndividualReward(uint256,uint256) (contracts/chef.sol#916-939) performs a multiplication on the result of a division:
        -sharePercentage = user_scope_1.totalUserBaseMul.mul(10000).div(poolBaseMul) (contracts/chef.sol#933-934)
        -user_scope_1.totalReward = user_scope_1.totalReward.add((_amount.mul(sharePercentage)).div(10000)) (contracts/chef.sol#935-937)
MasterChef.distributeFlatReward(uint256) (contracts/chef.sol#951-980) performs a multiplication on the result of a division:
        -sharePercentage = user_scope_3.totalUserBaseMul.mul(10000).div(allPoolBaseMul) (contracts/chef.sol#973-974)
        -user_scope_3.totalReward = user_scope_3.totalReward.add((_amount.mul(sharePercentage)).div(10000)) (contracts/chef.sol#975-977)
MasterChef.distributeTvlAdjustedReward(uint256) (contracts/chef.sol#992-1009) performs a multiplication on the result of a division:
        -poolRewardShare = tvl_scope_2.mul(10000).div(totalTvl) (contracts/chef.sol#1004)
        -reward = (_amount.mul(poolRewardShare)).div(10000) (contracts/chef.sol#1005)
MasterChef.updateCurBlockReward(MasterChef.UserInfo,uint256,uint256,address) (contracts/chef.sol#1173-1188) performs a multiplication on the result of a division:
        -sharePercentage = userBaseMul.mul(10000).div(totalPoolBaseMul) (contracts/chef.sol#1183)
        -currentUser.totalReward = currentUser.totalReward.add((totalBlockReward.mul(sharePercentage)).div(10000)) (contracts/chef.sol#1184-1186)
MasterChef.viewRewardInfo(uint256) (contracts/chef.sol#1194-1240) performs a multiplication on the result of a division:
        -sharePercentage = userBaseMul.mul(10000).div(totalPoolBaseMul) (contracts/chef.sol#1235)
        -currentUser.totalReward.add((totalBlockReward.mul(sharePercentage)).div(10000)) (contracts/chef.sol#1236-1239)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
MasterChef.calcstakingscore(uint256,uint256,uint256,uint256,uint256,uint256) (contracts/chef.sol#821-863) uses a dangerous strict equality:
        - vaultMonth == 12 (contracts/chef.sol#846)
MasterChef.getTodayReward(uint256) (contracts/chef.sol#1502-1515) uses a dangerous strict equality:
        - day == 0 (contracts/chef.sol#1507)
MasterChef.stakingScore(uint256,address) (contracts/chef.sol#765-809) uses a dangerous strict equality:
        - stkInfo.deposit == true (contracts/chef.sol#782)
MasterChef.updateBlockReward(uint256,address) (contracts/chef.sol#1110-1156) uses a dangerous strict equality:
        - lpSupply == 0 (contracts/chef.sol#1124)
MasterChef.updateClaimedReward(MasterChef.UserInfo,uint256) (contracts/chef.sol#1479-1493) uses a dangerous strict equality:
        - day == 0 (contracts/chef.sol#1484)
MasterChef.vaultMultiplier(uint256,address) (contracts/chef.sol#347-380) uses a dangerous strict equality:
        - stkInfo.deposit == true (contracts/chef.sol#361)
MasterChef.vaultMultiplier(uint256,address) (contracts/chef.sol#347-380) uses a dangerous strict equality:
        - stkInfo.vault == 12 (contracts/chef.sol#363)
MasterChef.vaultMultiplier(uint256,address) (contracts/chef.sol#347-380) uses a dangerous strict equality:
        - stkInfo.vault == 9 (contracts/chef.sol#365)
MasterChef.vaultMultiplier(uint256,address) (contracts/chef.sol#347-380) uses a dangerous strict equality:
        - stkInfo.vault == 6 (contracts/chef.sol#367)
MasterChef.viewRewardInfo(uint256) (contracts/chef.sol#1194-1240) uses a dangerous strict equality:
        - lpSupply == 0 (contracts/chef.sol#1200)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in MasterChef._withdraw(uint256,bool) (contracts/chef.sol#603-620):
        External calls:
        - withdrawASTRReward(_pid,_withStake) (contracts/chef.sol#607)
                - IERC20(ASTR).transfer(_to,_amount) (contracts/chef.sol#740)
        State variables written after the call(s):
        - _amount = checkEligibleAmount(_pid,msg.sender,true) (contracts/chef.sol#609)
                - stkInfo.deposit = false (contracts/chef.sol#691)
        - user.amount = user.amount.sub(_amount) (contracts/chef.sol#610)
        - user.rewardDebt = user.amount.mul(pool.accASTRPerShare).div(1e12) (contracts/chef.sol#611)
Reentrancy in MasterChef._withdraw(uint256,bool) (contracts/chef.sol#603-620):
        External calls:
        - withdrawASTRReward(_pid,_withStake) (contracts/chef.sol#607)
                - IERC20(ASTR).transfer(_to,_amount) (contracts/chef.sol#740)
        - pool.lpToken.safeTransfer(address(msg.sender),_amount) (contracts/chef.sol#612)
        State variables written after the call(s):
        - user.cooldown = false (contracts/chef.sol#614)
        - user.cooldowntimestamp = 0 (contracts/chef.sol#615)
        - user.totalUserBaseMul = 0 (contracts/chef.sol#616)
Reentrancy in MasterChef.deposit(uint256,uint256,uint256) (contracts/chef.sol#428-469):
        External calls:
        - pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (contracts/chef.sol#443-447)
        State variables written after the call(s):
        - staker.amount = _amount (contracts/chef.sol#458)
        - staker.totalAmount = user.amount (contracts/chef.sol#459)
        - staker.timestamp = block.timestamp (contracts/chef.sol#460)
        - staker.vault = vault (contracts/chef.sol#461)
        - staker.deposit = true (contracts/chef.sol#462)
        - user.amount = user.amount.add(_amount) (contracts/chef.sol#448)
        - user.timestamp = block.timestamp (contracts/chef.sol#465)
        - userStakingTrack[_pid][msg.sender] = userStakingTrack[_pid][msg.sender].add(1) (contracts/chef.sol#451-452)
Reentrancy in MasterChef.depositFromDAA(uint256,uint256,uint256,address,bool) (contracts/chef.sol#481-524):
        External calls:
        - pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (contracts/chef.sol#494-498)
        State variables written after the call(s):
        - staker.amount = _amount (contracts/chef.sol#513)
        - staker.totalAmount = user.amount (contracts/chef.sol#514)
        - staker.timestamp = block.timestamp (contracts/chef.sol#515)
        - staker.vault = vault (contracts/chef.sol#516)
        - staker.deposit = true (contracts/chef.sol#517)
        - user.amount = user.amount.add(_amount) (contracts/chef.sol#502)
        - user.timestamp = block.timestamp (contracts/chef.sol#520)
        - userStakingTrack[_pid][_sender] = userStakingTrack[_pid][_sender].add(1) (contracts/chef.sol#505-507)
Reentrancy in MasterChef.emergencyWithdraw(uint256) (contracts/chef.sol#700-708):
        External calls:
        - pool.lpToken.safeTransfer(address(msg.sender),_amount) (contracts/chef.sol#704)
        State variables written after the call(s):
        - user.amount = 0 (contracts/chef.sol#705)
        - user.totalReward = 0 (contracts/chef.sol#706)
Reentrancy in MasterChef.withdrawASTRReward(uint256,bool) (contracts/chef.sol#1353-1379):
        External calls:
        - slashExitFee(currentUser,_pid,dayCount) (contracts/chef.sol#1375)
                - IERC20(ASTR).transfer(_to,_amount) (contracts/chef.sol#740)
        State variables written after the call(s):
        - currentUser.totalReward = 0 (contracts/chef.sol#1378)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
```



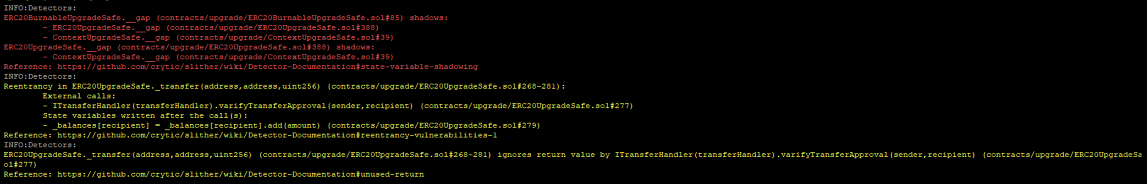
```
INFO:Detectors:
MasterChef.slashExitFee(MasterChef.UserInfo,uint256,uint256) (contracts/chef.sol#1447-1469) contains a tautology or contradiction:
        - fee < 0 (contracts/chef.sol#1456)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-contradiction
INFO:Detectors:
MasterChef.vaultMultiplier(uint256,address).vaultMul (contracts/chef.sol#353) is a local variable never initialized
MasterChef.getPremiumPayoutBonus(uint256,address).stakingscoreaddition (contracts/chef.sol#397) is a local variable never initialized
MasterChef.vaultMultiplier(uint256,address).depositCount (contracts/chef.sol#355) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
```

lm-pool.sol



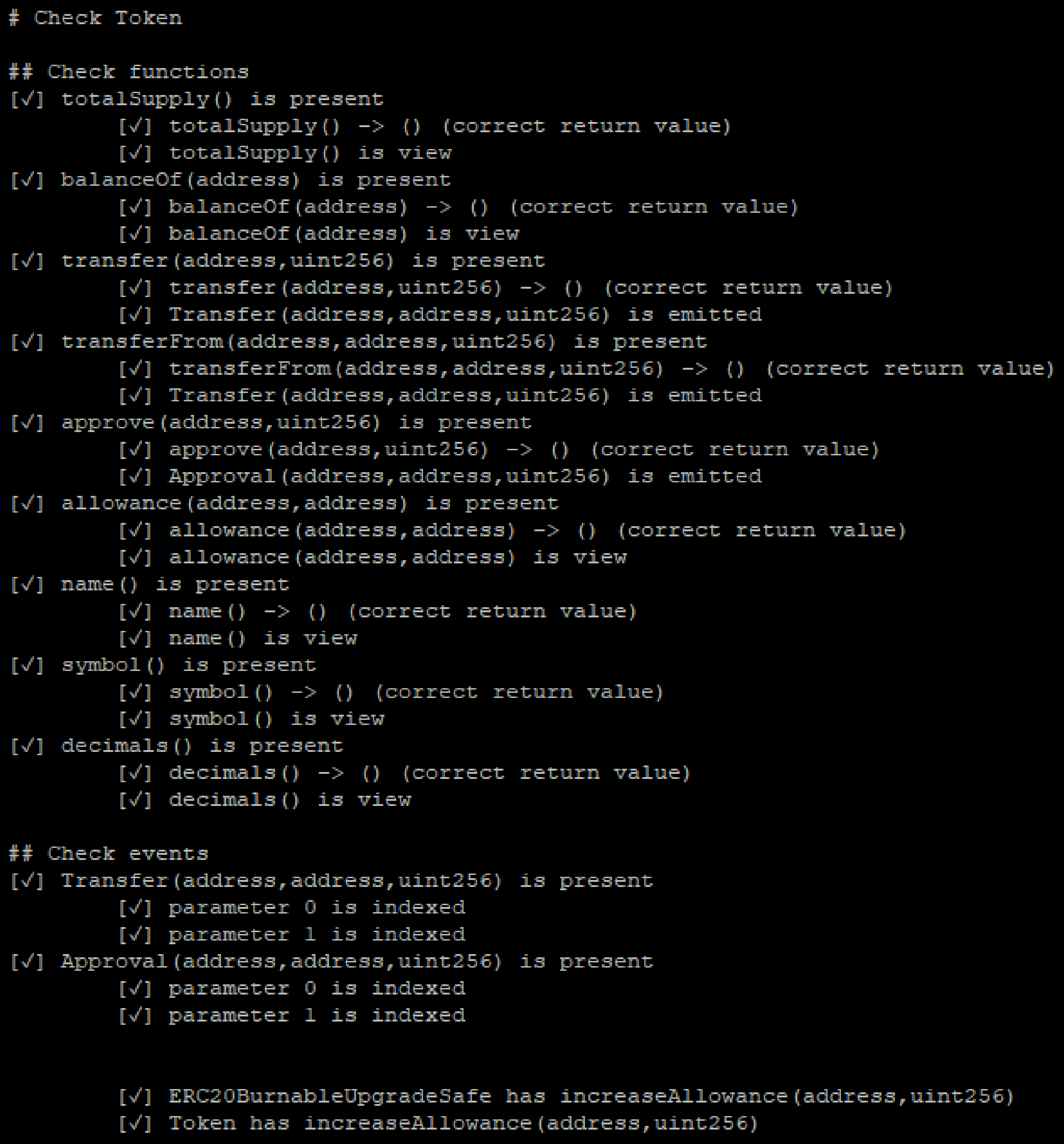
```
INFO:Detectors:
LmPool.updateClaimedReward(LmPool.UserInfo,uint256) (contracts/lm-pool.sol#865-879) uses a weak PRNG: "todayDaySeconds = block.timestamp % dayseconds (contracts/lm-pool.sol#876)"
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#weak-PRNG
INFO:Detectors:
LmPool.safeASTRTransfer(address,uint256) (contracts/lm-pool.sol#470-477) ignores return value by IERC20(ASTR).transfer(_to,_amount) (contracts/lm-pool.sol#476)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
LmPool.distributeIndividualReward(uint256,uint256) (contracts/lm-pool.sol#536-559) performs a multiplication on the result of a division:
        -sharePercentage = user_scope_1.totalUserBaseMul.mul(10000).div(poolBaseMul) (contracts/lm-pool.sol#553-554)
        -user_scope_1.totalReward = user_scope_1.totalReward.add((_amount.mul(sharePercentage)).div(10000)) (contracts/lm-pool.sol#555-557)
LmPool.distributeFlatReward(uint256) (contracts/lm-pool.sol#571-600) performs a multiplication on the result of a division:
        -sharePercentage = user_scope_3.totalUserBaseMul.mul(10000).div(allPoolBaseMul) (contracts/lm-pool.sol#593-594)
        -user_scope_3.totalReward = user_scope_3.totalReward.add((_amount.mul(sharePercentage)).div(10000)) (contracts/lm-pool.sol#595-597)
LmPool.distributeTvlAdjustedReward(uint256) (contracts/lm-pool.sol#612-629) performs a multiplication on the result of a division:
        -poolRewardShare = tvl_scope_2.mul(10000).div(totalTvl) (contracts/lm-pool.sol#624)
        -reward = (_amount.mul(poolRewardShare)).div(10000) (contracts/lm-pool.sol#625)
LmPool.updateCurBlockReward(LmPool.UserInfo,uint256,uint256) (contracts/lm-pool.sol#698-712) performs a multiplication on the result of a division:
        -sharePercentage = userBaseMul.mul(10000).div(totalPoolBaseMul) (contracts/lm-pool.sol#707)
        -currentUser.totalReward = currentUser.totalReward.add((totalBlockReward.mul(sharePercentage)).div(10000)) (contracts/lm-pool.sol#708-710)
LmPool.viewRewardInfo(uint256) (contracts/lm-pool.sol#718-764) performs a multiplication on the result of a division:
        -sharePercentage = userBaseMul.mul(10000).div(totalPoolBaseMul) (contracts/lm-pool.sol#759)
        -currentUser.totalReward.add((totalBlockReward.mul(sharePercentage)).div(10000)) (contracts/lm-pool.sol#760-763)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
LmPool.getTodayReward(uint256) (contracts/lm-pool.sol#888-901) uses a dangerous strict equality:
        - day == 0 (contracts/lm-pool.sol#893)
LmPool.updateBlockReward(uint256) (contracts/lm-pool.sol#641-682) uses a dangerous strict equality:
        - lpSupply == 0 (contracts/lm-pool.sol#655)
LmPool.updateClaimedReward(LmPool.UserInfo,uint256) (contracts/lm-pool.sol#865-879) uses a dangerous strict equality:
        - day == 0 (contracts/lm-pool.sol#870)
LmPool.viewRewardInfo(uint256) (contracts/lm-pool.sol#718-764) uses a dangerous strict equality:
        - lpSupply == 0 (contracts/lm-pool.sol#724)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in LmPool._withdraw(uint256,bool) (contracts/lm-pool.sol#339-353):
        External calls:
        - withdrawASTRReward(_pid,_withStake) (contracts/lm-pool.sol#343)
                - Chef(chefaddr).stakeASTRReward(msg.sender,_pid,_amount) (contracts/lm-pool.sol#814-818)
                - IERC20(ASTR).transfer(_to,_amount) (contracts/lm-pool.sol#476)
        State variables written after the call(s):
        - user.amount = user.amount.sub(_amount) (contracts/lm-pool.sol#346)
Reentrancy in LmPool._withdraw(uint256,bool) (contracts/lm-pool.sol#339-353):
        External calls:
        - withdrawASTRReward(_pid,_withStake) (contracts/lm-pool.sol#343)
                - Chef(chefaddr).stakeASTRReward(msg.sender,_pid,_amount) (contracts/lm-pool.sol#814-818)
                - IERC20(ASTR).transfer(_to,_amount) (contracts/lm-pool.sol#476)
        - pool.lpToken.safeTransfer(address(msg.sender),_amount) (contracts/lm-pool.sol#347)
        State variables written after the call(s):
        - user.cooldown = false (contracts/lm-pool.sol#349)
        - user.cooldowntimestamp = 0 (contracts/lm-pool.sol#350)
        - user.totalUserBaseMul = 0 (contracts/lm-pool.sol#351)
Reentrancy in LmPool.deposit(uint256,uint256,uint256) (contracts/lm-pool.sol#250-285):
        External calls:
        - pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (contracts/lm-pool.sol#261-265)
        State variables written after the call(s):
        - user.amount = user.amount.add(_amount) (contracts/lm-pool.sol#266)
        - user.timestamp = block.timestamp (contracts/lm-pool.sol#283)
        - userStakingTrack[_pid][msg.sender] = userStakingTrack[_pid][msg.sender].add(1) (contracts/lm-pool.sol#269-270)
Reentrancy in LmPool.emergencyWithdraw(uint256) (contracts/lm-pool.sol#433-441):
        External calls:
        - pool.lpToken.safeTransfer(address(msg.sender),_amount) (contracts/lm-pool.sol#437)
        State variables written after the call(s):
        - user.amount = 0 (contracts/lm-pool.sol#438)
        - user.totalReward = 0 (contracts/lm-pool.sol#439)
Reentrancy in LmPool.withdrawASTRReward(uint256,bool) (contracts/lm-pool.sol#780-803):
        External calls:
        - stakeASTRReward(Chef(chefaddr).ASTRPoolId(),_amount) (contracts/lm-pool.sol#788)
                - Chef(chefaddr).stakeASTRReward(msg.sender,_pid,_amount) (contracts/lm-pool.sol#814-818)
        - slashExitFee(currentUser,_pid,dayCount) (contracts/lm-pool.sol#799)
                - IERC20(ASTR).transfer(_to,_amount) (contracts/lm-pool.sol#476)
        State variables written after the call(s):
        - currentUser.totalReward = 0 (contracts/lm-pool.sol#802)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
LmPool.slashExitFee(LmPool.UserInfo,uint256,uint256) (contracts/lm-pool.sol#833-855) contains a tautology or contradiction:
        - fee < 0 (contracts/lm-pool.sol#842)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-contradiction
```

astr.sol



```
INFO:Detectors:
ERC20BurnableUpgradeSafe.__gap (contracts/upgrade/ERC20BurnableUpgradeSafe.sol#85) shadows:
        - ERC20UpgradeSafe.__gap (contracts/upgrade/ERC20UpgradeSafe.sol#388)
        - ContextUpgradeSafe.__gap (contracts/upgrade/ContextUpgradeSafe.sol#39)
ERC20UpgradeSafe.__gap (contracts/upgrade/ERC20UpgradeSafe.sol#388) shadows:
        - ContextUpgradeSafe.__gap (contracts/upgrade/ContextUpgradeSafe.sol#39)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
INFO:Detectors:
Reentrancy in ERC20UpgradeSafe._transfer(address,address,uint256) (contracts/upgrade/ERC20UpgradeSafe.sol#268-281):
        External calls:
        - ITransferHandler(transferHandler).varifyTransferApproval(sender,recipient) (contracts/upgrade/ERC20UpgradeSafe.sol#277)
        State variables written after the call(s):
        - _balances[recipient] = _balances[recipient].add(amount) (contracts/upgrade/ERC20UpgradeSafe.sol#279)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
ERC20UpgradeSafe._transfer(address,address,uint256) (contracts/upgrade/ERC20UpgradeSafe.sol#268-281) ignores return value by ITransferHandler(transferHandler).varifyTransferApproval(sender,recipient) (contracts/upgrade/ERC20UpgradeSa
ol#277)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

ERC CONFORMAL CHECKER FOR astr.sol passed:



```
# Check Token

## Check functions
[√] totalSupply() is present
        [√] totalSupply() -> () (correct return value)
        [√] totalSupply() is view
[√] balanceOf(address) is present
        [√] balanceOf(address) -> () (correct return value)
        [√] balanceOf(address) is view
[√] transfer(address,uint256) is present
        [√] transfer(address,uint256) -> () (correct return value)
        [√] Transfer(address,address,uint256) is emitted
[√] transferFrom(address,address,uint256) is present
        [√] transferFrom(address,address,uint256) -> () (correct return value)
        [√] Transfer(address,address,uint256) is emitted
[√] approve(address,uint256) is present
        [√] approve(address,uint256) -> () (correct return value)
        [√] Approval(address,address,uint256) is emitted
[√] allowance(address,address) is present
        [√] allowance(address,address) -> () (correct return value)
        [√] allowance(address,address) is view
[√] name() is present
        [√] name() -> () (correct return value)
        [√] name() is view
[√] symbol() is present
        [√] symbol() -> () (correct return value)
        [√] symbol() is view
[√] decimals() is present
        [√] decimals() -> () (correct return value)
        [√] decimals() is view

## Check events
[√] Transfer(address,address,uint256) is present
        [√] parameter 0 is indexed
        [√] parameter 1 is indexed
[√] Approval(address,address,uint256) is present
        [√] parameter 0 is indexed
        [√] parameter 1 is indexed


        [√] ERC20BurnableUpgradeSafe has increaseAllowance(address,uint256)
        [√] Token has increaseAllowance(address,uint256)
```

Slither yielded some positive results:

- Reentrancies: HAL08 - VIOLATION OF CHECK, EFFECTS, INTERACTIONS PATTERN
- Unchecked transfers: HAL04 - UNCHECKED TRANSFER
- Divide before multiply: HAL09 - DIVIDE BEFORE MULTIPLY
- Tautology expressions: HAL10 - TAUTOLOGY EXPRESSIONS

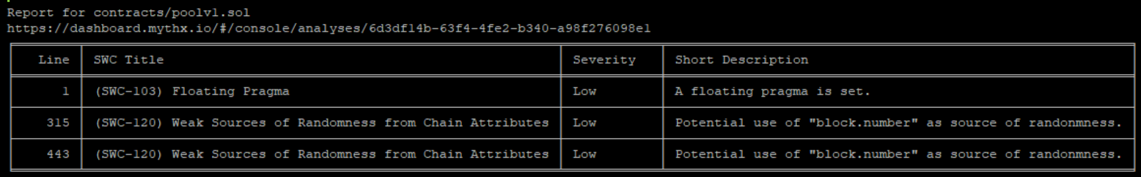
4.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on all the contracts and sent the compiled results to the analyzers to locate any vulnerabilities.

MythX results:

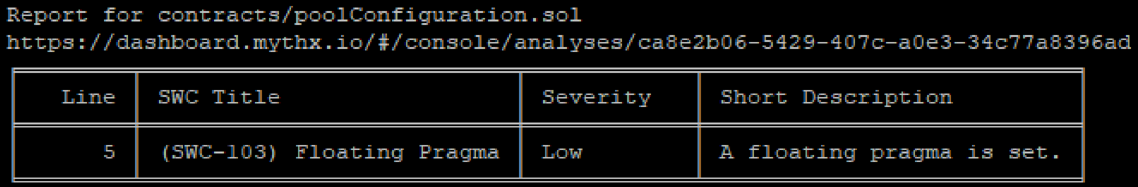
poolv1.sol



Report for contracts/poolv1.sol
https://dashboard.mythx.io/#/console/analyses/6d3df14b-63f4-4fe2-b340-a98f276098e1

Line	SWC Title	Severity	Short Description
1	(SWC-103) Floating Pragma	Low	A floating pragma is set.
315	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
443	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

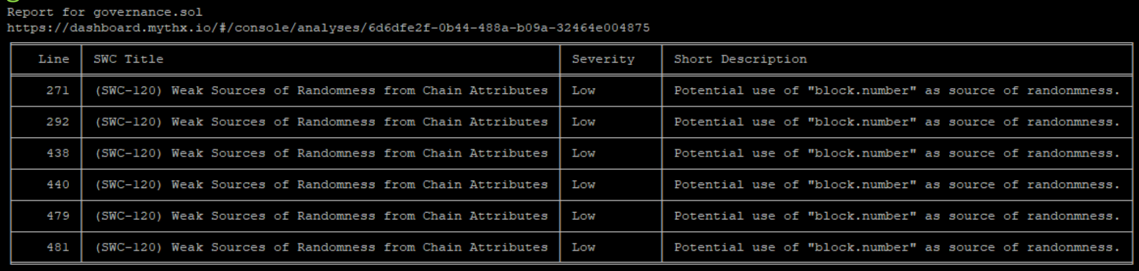
poolConfiguration.sol



Report for contracts/poolConfiguration.sol
https://dashboard.mythx.io/#/console/analyses/ca8e2b06-5429-407c-a0e3-34c77a8396ad

Line	SWC Title	Severity	Short Description
5	(SWC-103) Floating Pragma	Low	A floating pragma is set.

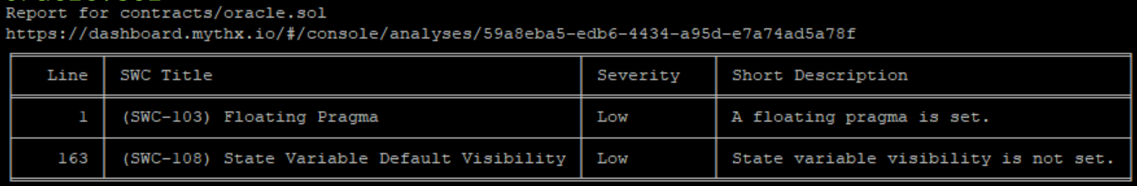
governance.sol



Report for governance.sol
https://dashboard.mythx.io/#/console/analyses/6d6dfe2f-0b44-488a-b09a-32464e004875

Line	SWC Title	Severity	Short Description
271	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
292	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
438	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
440	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
479	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
481	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

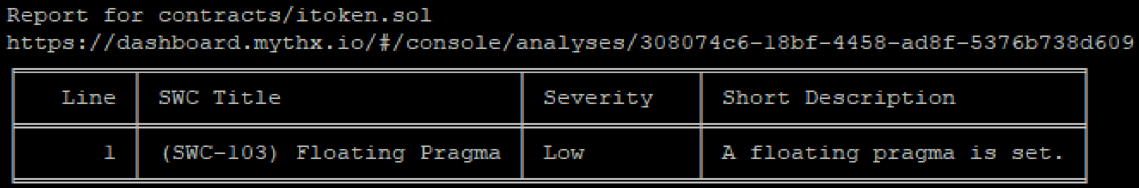
oracle.sol



Report for contracts/oracle.sol
https://dashboard.mythx.io/#/console/analyses/59a8eba5-edb6-4434-a95d-e7a74ad5a78f

Line	SWC Title	Severity	Short Description
1	(SWC-103) Floating Pragma	Low	A floating pragma is set.
163	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.

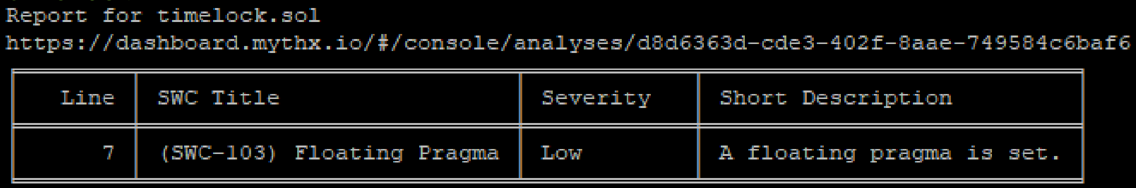
itoken.sol



Report for contracts/itoken.sol
https://dashboard.mythx.io/#/console/analyses/308074c6-18bf-4458-ad8f-5376b738d609

Line	SWC Title	Severity	Short Description
1	(SWC-103) Floating Pragma	Low	A floating pragma is set.

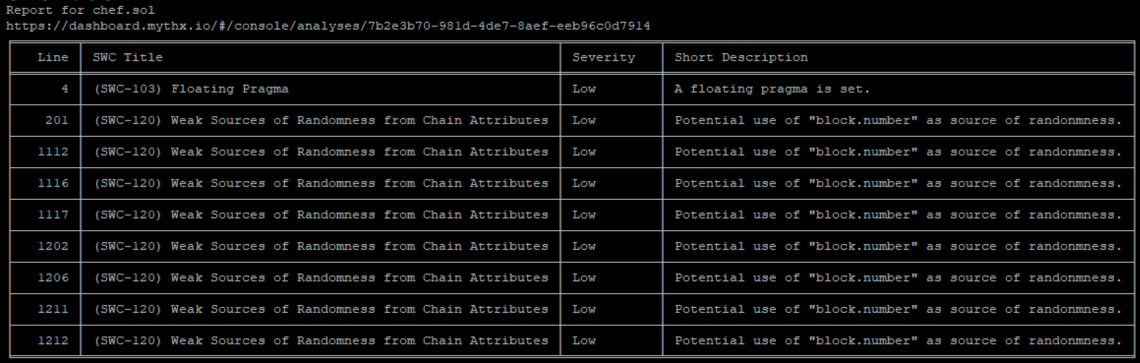
timelock



Report for timelock.sol
https://dashboard.mythx.io/#/console/analyses/d8d6363d-cde3-402f-8aae-749584c6baf6

Line	SWC Title	Severity	Short Description
7	(SWC-103) Floating Pragma	Low	A floating pragma is set.

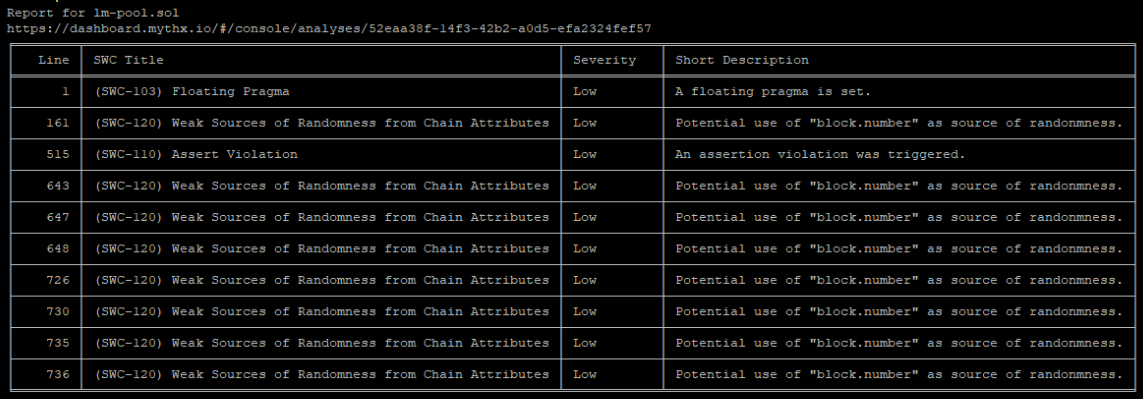
chef.sol



Report for chef.sol
https://dashboard.mythx.io/#/console/analyses/7b2e3b70-981d-4de7-8aef-eeb96c0d7914

Line	SWC Title	Severity	Short Description
4	(SWC-103) Floating Pragma	Low	A floating pragma is set.
201	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
1112	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
1116	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
1117	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
1202	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
1206	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
1211	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
1212	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

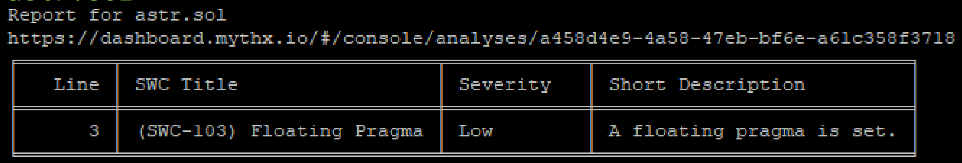
lm-pool.sol



Report for lm-pool.sol
https://dashboard.mythx.io/#/console/analyses/52eaa38f-14f3-42b2-a0d5-efa2324fef57

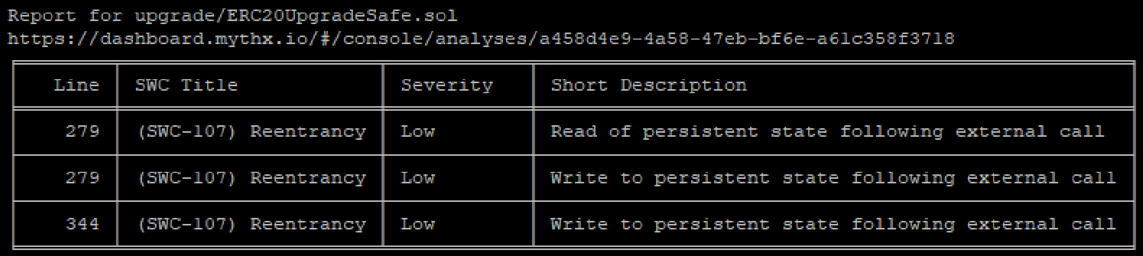
Line	SWC Title	Severity	Short Description
1	(SWC-103) Floating Pragma	Low	A floating pragma is set.
161	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
515	(SWC-110) Assert Violation	Low	An assertion violation was triggered.
643	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
647	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
648	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
726	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
730	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
735	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
736	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

astr.sol



Report for astr.sol
https://dashboard.mythx.io/#/console/analyses/a458d4e9-4a58-47eb-bf6e-a61c358f3718

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.



Report for upgrade/ERC20UpgradeSafe.sol
https://dashboard.mythx.io/#/console/analyses/a458d4e9-4a58-47eb-bf6e-a61c358f3718

Line	SWC Title	Severity	Short Description
279	(SWC-107) Reentrancy	Low	Read of persistent state following external call
279	(SWC-107) Reentrancy	Low	Write to persistent state following external call
344	(SWC-107) Reentrancy	Low	Write to persistent state following external call

No relevant findings came out from MythX. The assert violation in the contract lm-pool.sol is a false positive.

THANK YOU FOR CHOOSING

HALBORN