OBELISK

Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total: 90	2021-09-23	Zapmore, Donut	Audit Final

Audit Notes

Audit Date	2021-08-02 - 2021-09-20
Auditor/Auditors	Donut, DoD4uFN, Plemonade
Auditor/Auditors Contact Information	tibereum-obelisk@protonmail.com
Notes	Specified code and contracts are audited for security flaws. UI/UX (website), logic, team, and tokenomics are not audited.
Audit Report Number	OB588124582

Disclaimer

This audit is not financial, investment, or any other kind of advice and is for informational purposes only. This report is not a substitute for doing your own research and due diligence. Obelisk is not responsible or liable for any loss, damage, or otherwise caused by reliance on this report for any purpose. Obelisk has based this audit report solely on the information provided by the audited party and on facts that existed before or during the audit being conducted. Obelisk is not responsible for any outcome, including changes done to the contract/contracts after the audit was published. This audit is fully objective and only discerns what the contract is saying without adding any opinion to it. The audit is paid by the project but neither the auditors nor Obelisk has any other connection to the project and has no obligations other than to publish an objective report. Obelisk will always publish its findings regardless of the outcome of the findings. The audit only covers the subject areas detailed in this report and unless specifically stated, nothing else has been audited. Obelisk assumes that the provided information and material were not altered, suppressed, or misleading. This report is published by Obelisk, and Obelisk has sole ownership of this report. Use of this report for any reason other than for informational purposes on the subjects reviewed in this report including the use of any part of this report is prohibited without the express written consent of Obelisk.

Obelisk Auditing

Defi is a relatively new concept but has seen exponential growth to a point where there is a multitude of new projects created every day. In a fast-paced world like this, there will also be an enormous amount of scams. The scams have become so elaborate that it's hard for the common investor to trust a project, even though it could be legit. We saw a need for creating high-quality audits at a fast phase to keep up with the constantly expanding market. With the Obelisk stamp of approval, a legitimate project can easily grow its user base exponentially in a world where trust means everything. Obelisk Auditing consists of a group of security experts that specialize in security and structural operations, with previous work experience from among other things, PricewaterhouseCoopers. All our audits will always be conducted by at least two independent auditors for maximum security and professionalism.

As a comprehensive security firm, Obelisk provides all kinds of audits and project assistance.

Audit Information

The auditors always conducted a manual visual inspection of the code to find security flaws that automatic tests would not find. Comprehensive tests are also conducted in a specific test environment that utilizes exact copies of the published contract.

While conducting the audit, the Obelisk security team uses best practices to ensure that the reviewed contracts are thoroughly examined against all angles of attack. This is done by evaluating the codebase and whether it gives rise to significant risks. During the audit, Obelisk assesses the risks and assigns a risk level to each section together with an explanatory comment. Take note that the comments from the project team are their opinion and not the opinion of Obelisk.

Table of Contents

Version Notes 2

Audit Notes 2

Disclaimer 2

Obelisk Auditing 3

Audit Information 3

Project Information 6

Executive Summary 7

Summary Table 8

Introduction 11

Findings 12

Manual Analysis 12

Expired Assets Can Be Burned As If Not Expired 12

Reward Pools Can Be Created Without Matching Pairs 14

Unbound Loop 16

Modifying Local Copies Of Contract Variables 26

Swap For Fees Can Be Frontrun 27

Votes Are Not Reset After Vote Is Closed 28

Freezing And Ending Assets Can Be Voted On Multiple Times 32

Asset End Of Life Value Can Be Updated 34

Potential Overflow 36

Incompatibility With Uniswap 38

Deleted Array Members Are Not Removed 39

No Bounds When Setting Vesting Period 40

Protocol Values Should Be Public Or Have Getters 42

Assets Mintable After Expiry 44

Unusual Staked Amount Check 45

Harvesting Fees Assumes USDT Pair 46

No Way To Vote Against Asset End Of Life 47

Local Copies Of OpenZeppelin Contracts 48

Direct Modification Of ERC20 Values 51

Local Copies Of Uniswap Libraries 52

Incorrect Error Message 53

SafeMath Unnecessary In Solidity 0.8 54

Duplicate Contracts 55

Duplicate Contract Names 56

Unused Contract Values 57

Redundant Functions (Gas Optimization) 58

Duplicate Functionality 59

Interface Does Not Match Contract 60

Interface Is Empty 61

Unused Constructor Parameters 62
Address Indicates USDC Instead Of USDT 63
Struct Value Not Assigned 64
Function Does Not Burn Tokens 65
Rewards Are Lost If Not Collected Every Period 66
Rewards Labeled As Weekly But Calculated Daily 67
Maximum Vesting Schedules Vulnerability 68
Cannot Claim Rewards When Losing A Vote 70
Cannot Burn Tokens Marked Both Frozen And Expired 72
Static Analysis 74
Division Before Multiplication 74
No Events Emitted For Changes To Protocol Values 77
Missing Zero Checks 80
Unused Variables 82
Variable Can Be Declared Constant Or Immutable (Gas Optimization) 83
On-Chain Analysis 84
Unverified Contracts 84
Appendix A - Reviewed Documents 86
Appendix B - Risk Ratings 89
Appendix C - Testing Standard 89

Project Information

Project Name	ISSUAA
Description	First decentralized finance derivatives liquidity protocol without the need for over-collateralization empowers attractive returns at moderate risk for investors providing liquidity.
Website	https://www.issuaa.org/
Contact	@stockmaster0X
Contact information	@stockmaster0X on TG
Token Name(s)	ISSUAA Protocol Token
Token Short	IPT (refer to appendix A for asset tokens)
Contract(s)	See Appendix A
Code Language	Solidity
Chain	Polygon

Executive Summary

The audit of ISSUAA was conducted by three of Obelisks' security experts between the 2nd of August 2021 and the 20th of September 2021.

While combing through the provided code, the auditors found multiple issues with a mixture of severity. These issues were reported to the team and as the code was not deployed yet, the project team worked through and fixed all issues that could cause a serious adverse effect on the users. There is only one Unbound Loop finding left that is commented on by the team, please see the table and separate comment.

Other Informational findings are there for informational purposes and don't impact the project on a larger scale on audited implementation.

The team has not reviewed the UI/UX, logic, team, or tokenomics of the ISSUAA.

Please read the full document for a complete understanding of the audit.

Summary Table

Audited Part	ID	Severity	Resolved
Expired Assets Can Be Burned As If Not Expired	#0001	High Risk	Mitigated
Reward Pools Can Be Created Without Matching Pairs	#0002	High Risk	Mitigated
Unbound Loop	#0003	Medium Risk	See Comment
Modifying Local Copies Of Contract Variables	#0004	Medium Risk	Mitigated
Swap For Fees Can Be Frontrun	#0005	Medium Risk	Mitigated
Votes Are Not Reset After Vote Is Closed	#0006	Medium Risk	Mitigated
Freezing And Ending Assets Can Be Voted On Multiple Times	#0007	Medium Risk	Mitigated
Asset End Of Life Value Can Be Updated	#0008	Medium Risk	Mitigated
Potential Overflow	#0009	Low Risk	Mitigated
Incompatibility With Uniswap	#0010	Low Risk	Mitigated
Deleted Array Members Are Not Removed	#0011	Low Risk	Mitigated
No Bounds When Setting Vesting Period	#0012	Low Risk	Mitigated
Protocol Values Should Be Public Or Have Getters	#0013	Low Risk	Mitigated
Assets Mintable After Expiry	#0014	Low Risk	Mitigated
Unusual Staked Amount Check	#0015	Low Risk	Mitigated
Harvesting Fees Assumes USDT Pair	#0016	Low Risk	Mitigated
No Way To Vote Against Asset End Of Life	#0017	Low Risk	Mitigated
Local Copies Of OpenZeppelin Contracts	#0018	Informational	Mitigated

Direct Modification Of ERC20 Values	#0019	Informational	Mitigated
Local Copies Of Uniswap Libraries	#0020	Informational	Mitigated
Incorrect Error Message	#0021	Informational	Mitigated
SafeMath Unnecessary In Solidity 0.8	#0022	Informational	No
Duplicate Contracts	#0023	Informational	Mitigated
Duplicate Contract Names	#0024	Informational	Mitigated
Unused Contract Values	#0025	Informational	Mitigated
Redundant Functions (Gas Optimization)	#0026	Informational	Mitigated
Duplicate Functionality	#0027	Informational	Mitigated
Interface Does Not Match Contract	#0028	Informational	Mitigated
Interface Is Empty	#0029	Informational	Mitigated
Unused Constructor Parameters	#0030	Informational	Mitigated
Address Indicates USDC Instead Of USDT	#0031	Informational	Mitigated
Struct Value Not Assigned	#0032	Informational	Mitigated
Function Does Not Burn Tokens	#0033	Informational	Mitigated
Rewards Are Lost If Not Collected Every Period	#0034	Informational	See Comment
Rewards Labeled As Weekly But Calculated Daily	#0035	Informational	Mitigated
Division Before Multiplication	#0036	Medium Risk	Mitigated
No Events Emitted For Changes To Protocol Values	#0037	Informational	Partial
Missing Zero Checks	#0038	Informational	See Comment
Unused Variables	#0039	Informational	Mitigated

Variable Can Be Declared Constant Or Immutable (Gas Optimization)	#0040	Informational	Partial
Maximum Vesting Schedules Vulnerability	#0041	Medium Risk	Mitigated
Cannot Claim Rewards When Losing A Vote	#0042	Low Risk	Mitigated
Cannot Burn Tokens Marked Both Frozen And Expired	#0043	Low Risk	Mitigated
Unverified Contracts	#0044	Informational	Mitigated

Introduction

Obelisk was commissioned by ISSUAA on the 22nd of July 2021 to conduct a comprehensive audit of ISSUAAs' contracts. The following audit was conducted between the 2nd of August 2021 and the 20th of September 2021. Three of Obelisk's security experts went through the related contracts using industry standards to find if any vulnerabilities could be exploited.

The audit was conducted on contracts that were not yet live in a production environment. Working through the code, the auditors found multiple security risks in the provided contracts. All findings were handed over to the project team which worked fast and transparently on fixing the issues found. The auditors then re-audited the affected parts in order to conclude if the project team's fixes solved the issues found.

All severe issues found were worked on and all were solved by the team besides issue #3, "Unound Loop" in which the team provided a comment in which they state that the issue is highly unlikely to happen and what the consequences would be if it were to happen.

After the contracts were deployed, the auditors then made an on-chain analysis in order to make sure that the audited contracts matches the deployed contracts, which they did.

The informational findings are good to know while interacting with the project and most of the findings were mitigated. The informational findings that are left don't directly damage the project in its current state.

Please see each section of the audit to get a full understanding of the audit.

Findings

Manual Analysis

Expired Assets Can Be Burned As If Not Expired

SEVERITY	High Risk
RESOLVED	**YES**
FINDING ID	#0001
LOCATION	assetFactory.sol -> 245-263 assetFactory.sol -> 271-290

```
function burnAssets (
    string calldata _symbol,
    uint256 _amount
    )
    external
    {
    // ...
}
```

```
function burnExpiredAssets (
    string calldata _symbol,
    uint256 _amount1,
    uint256 _amount2
    )
    external
    {
    // ...
}
```

DESCRIPTION	The *burnAssets* function allows users to redeem tokens at any time. If the asset is frozen, the amount of long and short assets can be different. After a symbol is both frozen and expired, if the final price is lower than the maximum price, the short assets can be redeemed for the price difference. Then the long assets

	can be redeemed for the full original value. This allows users to redeem tokens for more than their stated value. The result will be a drain of the USDT balance of the assetFactory, preventing other users from redeeming their assets. Note: This exploit can be performed via a flash loan, in concert with the vote setting the asset as expired, draining the entire balance.
RECOMMENDATION	Prevent expired assets from being redeemed normally. Clarify when the frozen state is enabled and how that relates to the expired state.
MITIGATED/COMMENT	The vulnerability was resolved by restricting normal token burning to non-expired tokens.

Reward Pools Can Be Created Without Matching Pairs

SEVERITY	High Risk
RESOLVED	**YES**
FINDING ID	#0002
LOCATION	RewardMachine.sol -> 124-139

```
1     function addPools(
2         string calldata _symbol
3         )
4         external
5         onlyOwner
6         {
7         require(poolExists[_symbol] == false,'POOL_EXISTS_ALREADY');
8
   require(AssetFactory(assetFactoryAddress).assetExists(_symbol),'UNKNO
  WN_SYMBOL');
9         (address token1,address token2) =
  AssetFactory(assetFactoryAddress).getTokenAddresses(_symbol);
10        address pair1 =
  MarketFactory(marketFactoryAddress).getMarketPair(token1,USDTaddress);
11        address pair2 =
  MarketFactory(marketFactoryAddress).getMarketPair(token2,USDTaddress);
12        poolExists[_symbol] = true;
13        pools.push(pair1);
14        pools.push(pair2);
15        numberOfPools +=2;
16    }
```

DESCRIPTION	Pools can be added for a symbol that does not have associated pairs from the MarketFactory yet. This can result in the zero address being added multiple times to the pools list. All functionality related to collecting rewards will revert due to trying to call the 0 address.
RECOMMENDATION	Ensure that the pairs exist for any symbol which is added. This can be done by adding *required* statements to the function. Alternatively, automatically create pairs when creating the assets.
MITIGATED/COMMENT	The contract now checks that pairs exist for the asset tokens.

	Note: If pools are added per asset, this will only allow 124 assets to be added.

Unbound Loop

SEVERITY	Medium Risk
RESOLVED	**SEE COMMENT**
FINDING ID	#0003
LOCATION	DAO.sol -> 225-229 DAO.sol -> 232-235 DAO.sol -> 237-240 DAO.sol -> 381-385 DAO.sol -> 387-390 DAO.sol -> 392-395 GovernanceToken.sol -> 143-145 GovernanceToken.sol -> 204-206 GovernanceToken.sol -> 242-244 GovernanceToken.sol -> 275-277 GovernanceToken.sol -> 315-317 GovernanceToken.sol -> 334-337 MarketERC20.sol -> 66-77 RewardMachine.sol -> 171-174 RewardMachine.sol -> 207-224 RewardMachine.sol -> 262-280 VoteMachine.sol -> 250-253 VoteMachine.sol -> 256-259 VoteMachine.sol -> 261-264 VoteMachine.sol -> 344-354 VoteMachine.sol -> 356-359 VoteMachine.sol -> 377-379

```
            for (uint256 i = 0; i < getGrantVotes[_receiver].voteNumber; i++) {
             voteMachine.addRewardPointsDAO(getGrantVotes[_receiver].individualVotes[i].votingAddress, getGrantVotes[_receiver].individualVotes[i].yesVotes);
                newRewardpoints = newRewardpoints.add(getGrantVotes[_receiver].individualVotes[i].yesVotes);
                }
            }
```

```
           for (uint256 i = 0; i < getGrantVotes[_receiver].voteNumber; i++) {
             voteMachine.addRewardPointsDAO(getGrantVotes[_receiver].individualVotes[i].votingAddress, getGrantVotes[_receiver].individualVotes[i].noVotes);
                newRewardpoints = newRewardpoints.add(getGrantVotes[_receiver].individualVotes[i].noVotes);
            }
```

```
        for (uint256 i = 0; i < getGrantVotes[_receiver].voteNumber; i++) {
            address voteAddress = getGrantVotes[_receiver].individualVotes[i].votingAddress;
            delete(getGrantVotes[_receiver].hasvoted[voteAddress]);
        }
```

```
            for (uint256 i = 0; i < getNewAssetVotes[_symbol].voteNumber; i++) {
             voteMachine.addRewardPointsDAO(getNewAssetVotes[_symbol].individualVotes[i].votingAddress, getNewAssetVotes[_symbol].individualVotes[i].yesVotes);
                newRewardpoints = newRewardpoints.add(getNewAssetVotes[_symbol].individualVotes[i].yesVotes);
                }
            }
```

```
            for (uint256 i = 0; i < getNewAssetVotes[_symbol].voteNumber; i++) {
             voteMachine.addRewardPointsDAO(getNewAssetVotes[_symbol].individualVotes[i].votingAddress, getNewAssetVotes[_symbol].individualVotes[i].noVotes);
                newRewardpoints = newRewardpoints.add(getNewAssetVotes[_symbol].individualVotes[i].noVotes);
            }
```

```
        for (uint256 i = 0; i < getGrantVotes[_receiver].voteNumber; i++) {
            address voteAddress = getGrantVotes[_receiver].individualVotes[i].votingAddress;
            delete(getGrantVotes[_receiver].hasvoted[voteAddress]);
        }
```

```
    for (uint256 s = 0; s < stakeholders.length; s += 1){
        if (_address == stakeholders[s]) return (true, s);
     }
```

```
    for (uint256 s = 0; s < stakeholders.length; s += 1){
        _totalStakes = _totalStakes.add(stakes[stakeholders[s]]);
    }
```

```
    for (uint256 i = 0; i < vestingSchedules[msg.sender].length; i += 1){
        if(vestingSchedules[msg.sender][i][0] < block.timestamp) {delete vestingSchedules[msg.sender][i];}
    }
```

```
        for (uint256 i=0; i < schedule.length;i++){
          if (schedule[i][0] > block.timestamp) {vestedStake = vestedStake.add(schedule[i][1]);}
          }
```

```
    for (uint256 i=0; i < schedule.length;i++){
      if (schedule[i][0] > _time) {lockedStake = lockedStake.add(schedule[i][1]);}
      }
```

```
    for (uint256 i=0; i>schedule.length;i++){
      if (schedule[i][0] > block.timestamp) {lockedStake = lockedStake.add(schedule[i][1]);}
      else {delete schedule[i];}
    }
```

```
    function isHolder(
        address _address
        )
        public
        view
        returns(bool, uint256)
        {
        for (uint256 s = 0; s < holders.length; s += 1){
           if (_address == holders[s]) return (true, s);
        }
        return (false, 0);
    }
```

```
        for (uint256 i = 0; i < getNewAssetVotes[_symbol].voteNumber;
i++) {
            address voteAddress =
getNewAssetVotes[_symbol].individualVotes[i].votingAddress;
            delete(getNewAssetVotes[_symbol].hasvoted[voteAddress]);
        }
```

```
            for (uint256 s = 0; s < numberOfPools; s += 1){
                // ...
            }
```

```
            for (uint256 s = 0; s < numberOfPools; s += 1){
                // ...
            }
```

```
            for (uint256 i = 0; i < getFreezeVotes[_symbol].voteNumber; i++) {
             addRewardPoints(getFreezeVotes[_symbol].individualVotes[i].votingAddress, getFreezeVotes[_symbol].individualVotes[i].yesVotes);
                newRewardpoints = newRewardpoints.add(getFreezeVotes[_symbol].individualVotes[i].yesVotes);
                }
            }
```

```
            for (uint256 i = 0; i < getFreezeVotes[_symbol].voteNumber; i++) {
             addRewardPoints(getFreezeVotes[_symbol].individualVotes[i].votingAddress, getFreezeVotes[_symbol].individualVotes[i].noVotes);
                newRewardpoints = newRewardpoints.add(getFreezeVotes[_symbol].individualVotes[i].noVotes);
            }
```

```
        for (uint256 i = 0; i < getFreezeVotes[_symbol].voteNumber; i++) {
            address voteAddress = getFreezeVotes[_symbol].individualVotes[i].votingAddress;
            delete(getFreezeVotes[_symbol].hasvoted[voteAddress]);
        }
```

```
        for (uint256 i = 0; i < getEndOfLifeVotes[_symbol].voteNumber; i++) {
            // ...
        }
```

```
        for (uint256 i = 0; i < getEndOfLifeVotes[_symbol].voteNumber; i++) {
            address voteAddress = getEndOfLifeVotes[_symbol].individualVotes[i].votingAddress;
            delete(getEndOfLifeVotes[_symbol].hasvoted[voteAddress]);
        }
```

```
        for (uint256 s = 0; s < rewardPointsSnapshots[currentRewardsRound].votingRewardAddresses.length; s += 1){
            if (_address == rewardPointsSnapshots[currentRewardsRound].votingRewardAddresses[s]) return (true, s);
        }
```

LOCATION	Revision-2: VoteMachine.sol -> 365-374 VoteMachine.sol -> 390-419 VoteMachine.sol -> 536-544 VoteMachine.sol -> 614-633

```
        for (uint256 s = 0; s < freezeVotesToCheck[_rewardsRound][_address].length; s += 1){
            // ...
        }
```

```
        uint256 numberOfVotesToCheck = freezeVotesToCheck[_rewardsRound][_address].length;
        for (uint256 s = 0; s < numberOfVotesToCheck; s += 1){
            // ...
        }

        numberOfVotesToCheck = expiryVotesToCheck[_rewardsRound][_address].length;
        for (uint256 s = 0; s < numberOfVotesToCheck; s += 1){
            // ...
        }
```

```
        for (uint256 s = 0; s < expiryVotesToCheck[_rewardsRound][_address].length; s += 1){
            // ...
        }
```

```
        uint256 numberOfVotesToCheck =
freezeVotesToCheck[currentRewardsRound
-1][_address].length;
        for (uint256 s = 0; s < numberOfVotesToCheck; s +=
1){
            // ...
        }

        numberOfVotesToCheck =
expiryVotesToCheck[currentRewardsRound
-1][_address].length;
        for (uint256 s = 0; s < numberOfVotesToCheck; s +=
1){
            // ...
        }
```

DESCRIPTION	Iterating over an unbounded array can cause transactions to revert due to the gas limit.
RECOMMENDATION	Use mappings and other techniques to remove loops. Prevent the number of iterations from exceeding a pre-determined number.
MITIGATED/COMMENT	All previously noted loops were either removed or changed to limit the number of iterations. VoteMachine's algorithm was changed to use a different loop which may be potentially unbound. Project team comment: “The current implementation includes an unbound loop when the users claim their weekly rewards. Under normal circumstances, it seems highly unlikely that a user will reach the number of voting processes that are necessary to make this call fail. Expiry votes can only be done once a year and all other votes require a threshold of staked IPT tokens, which means that only larger holders of IPT can trigger them. This should also help o avoid having a number of voting processes that is inflated to a level where this could cause a problem. If this should happen, however, the risk is limited to only one week of rewards.”

	Obelisk comment: “A user could potentially participate in every vote. Ensure that the number of possible votes will not cause rewards to be locked.”

Modifying Local Copies Of Contract Variables

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0004
LOCATION	GovernanceToken.sol -> 332 GovernanceToken.sol -> 358

```
1     uint256[2][] memory schedule = vestingSchedule(_address);
2
```

DESCRIPTION	The value of the schedule is modified locally in these functions: GovernanceToken.lockStake GovernanceToken.lockStakeForVote These changes will not be saved after the function exits.
RECOMMENDATION	Use the *storage* keyword to modify values on the chain.
MITIGATED/COMMENT	Modification of local copies is removed.

Swap For Fees Can Be Frontrun

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0005
LOCATION	assetFactory.sol -> 366

```
1         uint256[] memory amounts =
  IMarketRouter02(marketRouterAddress).swapExactTokensForTokens(tokenAmt
  ,0,path,address(this),block.timestamp.add(1 hours));
2
```

DESCRIPTION	The *assetFactory* contract trades for USDT on the router but does not provide a slippage limit. The rewards from the swap can be front-run. This function is accessible to all users. As a result, it is vulnerable to sandwich attacks.
RECOMMENDATION	Provide a slippage limit for the token as a parameter or use an oracle's time-weighted average price (TWAP) in order to prevent frontrunning.
MITIGATED/COMMENT	Project team comment: “By limiting the price impact the number of tokens to be harvested to 0.6% of the USDC pool I am effectively implementing that the price impact of this is not more than 0.6%. However, for a sandwich attack to be profitable, the price impact will need to be higher, as the attacker would have to pay 0.3% in trading fees twice. In order to make it impossible to call the function several times in order to avoid this check, there is a cooldown implemented.” Obelisk comment: “While unconventional, this method should limit the risk of front-running reward swaps.”

Votes Are Not Reset After Vote Is Closed

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0006
LOCATION	DAO.sol -> 222-240 DAO.sol -> 376-395

```
        if (getGrantVotes[_receiver].yesVotes > getGrantVotes[_receiver].noVotes){

         governanceToken.transfer(_receiver,getGrantVotes[_receiver].amount);

            for (uint256 i = 0; i < getGrantVotes[_receiver].voteNumber; i++) {

             voteMachine.addRewardPointsDAO(getGrantVotes[_receiver].individualVotes[i].votingAddress, getGrantVotes[_receiver].individualVotes[i].yesVotes);
                newRewardpoints = newRewardpoints.add(getGrantVotes[_receiver].individualVotes[i].yesVotes);
                }
            }
        else {
            DAOVolume = DAOVolume.add(getGrantVotes[_receiver].amount);
            for (uint256 i = 0; i < getGrantVotes[_receiver].voteNumber; i++) {

             voteMachine.addRewardPointsDAO(getGrantVotes[_receiver].individualVotes[i].votingAddress, getGrantVotes[_receiver].individualVotes[i].noVotes);
                newRewardpoints = newRewardpoints.add(getGrantVotes[_receiver].individualVotes[i].noVotes);
            }
        }
        for (uint256 i = 0; i < getGrantVotes[_receiver].voteNumber; i++) {
            address voteAddress = getGrantVotes[_receiver].individualVotes[i].votingAddress;
            delete(getGrantVotes[_receiver].hasvoted[voteAddress]);
        }
```

```
1         if (getNewAssetVotes[_symbol].yesVotes >
  getNewAssetVotes[_symbol].noVotes){
2             // HIER DAS REIN WAS ER MACHEN SOLL
3
   assetFactory.createAssets(getNewAssetVotes[_symbol].name,_symbol,getN
  ewAssetVotes[_symbol].description,getNewAssetVotes[_symbol].upperLimit
  );
4
5
6             for (uint256 i = 0; i <
  getNewAssetVotes[_symbol].voteNumber; i++) {
7
   voteMachine.addRewardPointsDAO(getNewAssetVotes[_symbol].individualVo
  tes[i].votingAddress,
  getNewAssetVotes[_symbol].individualVotes[i].yesVotes);
8                 newRewardpoints =
  newRewardpoints.add(getNewAssetVotes[_symbol].individualVotes[i].yesVo
  tes);
9                 }
10            }
11        else {
12            for (uint256 i = 0; i <
  getNewAssetVotes[_symbol].voteNumber; i++) {
13
   voteMachine.addRewardPointsDAO(getNewAssetVotes[_symbol].individualVo
  tes[i].votingAddress,
  getNewAssetVotes[_symbol].individualVotes[i].noVotes);
14                newRewardpoints =
  newRewardpoints.add(getNewAssetVotes[_symbol].individualVotes[i].noVot
  es);
15            }
16        }
17        for (uint256 i = 0; i < getNewAssetVotes[_symbol].voteNumber;
  i++) {
18            address voteAddress =
  getNewAssetVotes[_symbol].individualVotes[i].votingAddress;
19            delete(getNewAssetVotes[_symbol].hasvoted[voteAddress]);
20        }
```

DESCRIPTION	The *individualVotes* of each vote are not reset, they are closed. Voters can be issued rewards for past votes. A vote can be reopened with past votes for the following conditions: - Grantees can receive multiple votes - A new asset symbol that is not created can be voted for again As votes are not reset, the likelihood of the past vote result passing again increases.
RECOMMENDATION	Ensure that past votes are not included in voting rewards.

MITIGATED/COMMENT	New logic for voting was implemented, resolving this issue.

Freezing And Ending Assets Can Be Voted On Multiple Times

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0007
LOCATION	VoteMachine.sol -> 148-161 VoteMachine.sol -> 274-289 VoteMachine.sol -> 236-286 VoteMachine.sol -> 327-362

```
1     function initiateFreezeVote(
2         string calldata _symbol
3         )
4         external
5         {
6         require (assetFactory.assetExists(_symbol),'ASSET_UNKNOWN');
  //check if the symbol already exists
7         require (getFreezeVotes[_symbol].open ==
  false,'VOTE_IS_OPEN');   //check if the voting process is open
8         // ...
9     }
```

```
1     function initiateEndOfLifeVote(
2         string calldata _symbol
3         )
4         external
5         {
6         require (assetFactory.assetExists(_symbol),'ASSET_UNKNOWN');
  //check if the symbol already exists
7         require (getEndOfLifeVotes[_symbol].open ==
  false,'VOTE_OPEN');
8         require(assetFactory.getExpiryPrice(_symbol) <
  block.timestamp, 'ASSET_NOT_EXPIRED');
9         // ...
10        }
```

```
function closeFreezeVote (
    string calldata _symbol
    )
    external
    {
    // ...
    delete(getFreezeVotes[_symbol]);

}
```

```
function closeEndOfLifeVote (
    string calldata _symbol
    )
    external
    {
    // ...
    delete(getEndOfLifeVotes[_symbol]);
}
```

DESCRIPTION	Vote machine deletes records when closing its votes. As a result, the votes can be opened again with no restriction. Malicious actors can use this to collect rewards by continuously re-opening past votes.
RECOMMENDATION	Prevent old votes from being reopened. This can be done by checking that the asset records have been modified correctly. For example, checking that an asset has been frozen or not in the *initiateFreezeVote*() function.
MITIGATED/COMMENT	Initiating a vote to freeze already frozen assets and expire already expired assets is now prevented.

Asset End Of Life Value Can Be Updated

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0008
LOCATION	assetFactory.sol -> 328-337 VoteMachine.sol -> 274-289 VoteMachine.sol -> 327-362

```
    function setEndOfLifeValue(
        string calldata _symbol,
        uint256 _value
        )
        external
        {
        require(msg.sender == voteMachineAddress);
        getAsset[_symbol].endOfLifeValue = _value;
        getAsset[_symbol].expired = true;
    }
```

```
  function initiateEndOfLifeVote(
        string calldata _symbol
        )
        external
        {
        require (assetFactory.assetExists(_symbol),'ASSET_UNKNOWN');
//check if the symbol already exists
        require (getEndOfLifeVotes[_symbol].open ==
false,'VOTE_OPEN');
        require(assetFactory.getExpiryPrice(_symbol) <
block.timestamp, 'ASSET_NOT_EXPIRED');
        // ...
        }
```

```
1    function closeEndOfLifeVote (
2         string calldata _symbol
3         )
4         external
5         {
6         // ...
7         assetFactory.setEndOfLifeValue(_symbol,endOfLiveValue);
8
9         // ...
10        delete(getEndOfLifeVotes[_symbol]);
11    }
```

DESCRIPTION	The end-of-life vote can be called multiple times if it is set to a low value. Each time it is called, the value can be changed.
RECOMMENDATION	Prevent the end-of-life value from changing.
MITIGATED/COMMENT	End-of-life voting changed so that it cannot be called multiple times.

Potential Overflow

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0009
LOCATION	MarketPair.sol -> 134-159

```
1     function _mintFee(
2         uint256 _reserve0,
3         uint256 _reserve1
4         )
5         private
6         returns (bool feeOn)
7         {
8         address feeTo = IMarketFactory(factory).feeTo();
9         feeOn = feeTo != address(0);
10        uint256 _kLast = kLast; // gas savings
11        if (feeOn) {
12            if (_kLast != 0) {
13                uint256 rootK =
  Math.sqrtu(uint(_reserve0).mul(_reserve1));
14                uint256 rootKLast = Math.sqrtu(_kLast);
15                if (rootK > rootKLast) {
16                    uint256 numerator =
  totalSupply().mul(rootK.sub(rootKLast));
17                    uint256 denominator = rootK.mul(5).add(rootKLast);
18                    uint256 liquidity = (numerator) / denominator;
19                    if (liquidity > 0) _mint(feeTo, liquidity);
20                }
21            }
22        } else if (_kLast != 0) {
23            kLast = 0;
24        }
25        return feeOn;
26    }
```

DESCRIPTION	Minting multiplies two uint256 values, which can potentially overflow. This will cause all of the functions to revert if the *MarketFactory.feeTo* address is non-zero and the product of the reserves is very high. However, if the *feeTo* value is set to 0 before adding liquidity, a pair with high reserves of both tokens will be permanently locked.
RECOMMENDATION	Use the default Uniswap type of uint112 for reserves.

MITIGATED/COMMENT	Reserves are now stored as uint112.

Incompatibility With Uniswap

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0010
LOCATION	MarketPair.sol -> 24-26

```
1     uint256 private reserve0;           // uses single storage slot,
  accessible via getReserves
2     uint256 private reserve1;           // uses single storage slot,
  accessible via getReserves
3     uint32  private blockTimestampLast;
```

DESCRIPTION	Reserve value data types are changed to uint256 from Uniswap's original uint112. The contract also does not track the cumulative prices. These changes result in incompatibility with standard uniswap libraries. Other projects may find it difficult, and possibly impossible to implement functionality based on this exchange (eg. an oracle).
RECOMMENDATION	Revert the changes to match Uniswap.
MITIGATED/COMMENT	Project team comment: “With regard to tracking prices: we currently see no need for other protocols to use our prices as an oracle”

Deleted Array Members Are Not Removed

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0011
LOCATION	GovernanceToken.sol -> 243 GovernanceToken.sol -> 336 GovernanceToken.sol -> 362

```
1         if(vestingSchedules[msg.sender][i][0] < block.timestamp)
  {delete vestingSchedules[msg.sender][i];}
2
```

```
1       else {delete schedule[i];}
2
```

DESCRIPTION	Deleted array members are not removed from the array. This leaves [0,0] entries throughout the *vestingSchedule* array. These can later be modified by other schedule updating functions such as *setMinimumVestingPeriod*.
RECOMMENDATION	Ensure that deleted array members are properly removed from the array. A typical way to do this is to copy the last array member to the deleted index and pop the array.
MITIGATED/COMMENT	Noted delete statements were removed.

No Bounds When Setting Vesting Period

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0012
LOCATION	GovernanceToken.sol -> 287-297 GovernanceToken.sol -> 351-368

```
1   function setMinimumVestingPeriod(
2     address _address,
3     uint256 _timestamp
4     )
5     internal
6     {
7     uint256[2][] memory schedule = vestingSchedule(_address);
8     for (uint256 i=0; i < schedule.length;i++){
9       if (schedule[i][0] < _timestamp) {vestingSchedules[_address][i]
  [0] = _timestamp;}
10      }
11    }
```

```
1   function lockStakeForVote(
2     address _address,
3     uint256 _timestamp
4     )
5     external
6     {
7     require (msg.sender == voteMachineAddress || msg.sender ==
  DAOAddress,"NOT_VM_ADRESS");
8     uint256[2][] memory schedule = vestingSchedule(_address);
9     uint256 lockedStake = 0;
10    for (uint256 i=0; i>schedule.length;i++){
11      if (schedule[i][0] > block.timestamp) {lockedStake =
  lockedStake.add(schedule[i][1]);}
12      else {delete schedule[i];}
13    }
14    uint256  currentStake = stakeOf(msg.sender);
15    uint256  unlockedStake = currentStake.sub(lockedStake);
16    vestingSchedules[_address].push([_timestamp,unlockedStake]);
17    setMinimumVestingPeriod(_address,_timestamp);
18  }
```

DESCRIPTION	The function *setMinimumVestingPeriod* sets a user's vesting

	schedule. Since there are no checks to how far this timestamp is from the present, the vesting can last forever. *setMinimumVestingPeriod* is being used by *lockStakeForVote* and neither of them checks the *_timestamp*. A bad actor could lock away user funds forever. NOTE: We will confirm that the voteMachine/DAO has reasonable limits.
RECOMMENDATION	Add a maximum amount of time which vesting period will last.
MITIGATED/COMMENT	Added a 2-year maximum vesting delay.

Protocol Values Should Be Public Or Have Getters

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0013
LOCATION	assetFactory.sol -> 25-30 DAO.sol -> 15-17 GovernanceToken.sol -> 10-11 RewardMachine.sol -> 14-16 RewardMachine.sol -> 29-31 RewardMachine.sol -> 38 VoteMachine.sol -> 69

```
1     address tokenFactoryAddress;
2     address voteMachineAddress;
3     address rewardsMachineAddress;
4     address marketFactoryAddress;
5     address marketRouterAddress;
6     address INTAddress;
```

```
1    address voteMachineAddress;
2     address rewardsMachineAddress;
3     address assetFactoryAddress;
```

```
1   address voteMachineAddress;
2   address DAOAddress;
```

```
1     address assetFactoryAddress;
2     address rewardsMachineAddress;
3     address DAOAddress;
```

```
1     address voteMachineAddress;
2     address assetFactoryAddress;
3     address marketFactoryAddress;
```

```
1     bool IPTBonusPoolAdded = false;
2
```

```
1     mapping (uint256 => rewardPointsSnapshot) internal
  rewardPointsSnapshots;
```

DESCRIPTION	Important values should have getters so that they can be easily inspected on-chain.
RECOMMENDATION	Add getter functions or change the values to be public.
MITIGATED/COMMENT	Values were changed to public.

Assets Mintable After Expiry

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0014
LOCATION	assetFactory.sol -> 226-238

```
1     function mintAssets (
2         string calldata _symbol,
3         uint256 _amount
4         )
5         external
6         {
7         require (getAsset[_symbol].frozen == false &&
  getAsset[_symbol].expiryTime > block.timestamp,'INVALID');
8
   IERC20(USDTaddress).transferFrom(msg.sender,address(this),_amount);
9         uint256 USDDecimals = ERC20(USDTaddress).decimals();
10        uint256 tokenAmount = _amount.mul(10**(18-
  USDDecimals)).div(getAsset[_symbol].upperLimit/1000);
11
   TokenFactory(tokenFactoryAddress).mint(getAsset[_symbol].Token1,
  msg.sender, tokenAmount);
12
   TokenFactory(tokenFactoryAddress).mint(getAsset[_symbol].Token2,
  msg.sender, tokenAmount);
13    }
```

DESCRIPTION	Assets can be minted if the symbol is marked as expired under the following conditions: - The asset's expiry time has not passed - The asset was not marked as frozen.
RECOMMENDATION	Clarify the relationship between *issuaaLibrary.Asset.expiryTime* and *issuaaLibrary.Asset.expired*. It may be necessary to add another *require* check to *AssetFactory.setEndOfLifeValue()*.
MITIGATED/COMMENT	Project team comment: “As all assets which have *issuaaLibrary.Asset.expired* == true is also past that date, no expired assets can be minted.”

Unusual Staked Amount Check

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0015
LOCATION	GovernanceToken.sol -> 339-340

```
1     uint256  unlockedStake = currentStake.div(lockedStake);
2     require (unlockedStake >= _stake,'Not enough free stake
  available');
```

DESCRIPTION	The unlocked stake is checked by dividing the current stake with the locked stake. This is likely incorrect.
RECOMMENDATION	Confirm that this is the expected behavior.
MITIGATED/COMMENT	The division was replaced with subtraction.

Harvesting Fees Assumes USDT Pair

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0016
LOCATION	assetFactory.sol -> 357-359

```
1         uint256 tokenAmt = token0 == USDTaddress ? amount1 : amount0;
2         uint256 usdAmt = token0 == USDTaddress ? amount0 : amount1;
3         address tokenAddress = token0 == USDTaddress ? token1 :
  token0;
```

DESCRIPTION	The liquidity pair used to collect fees is assumed to have USDT as one of its tokens. This is not explicitly enforced. As a result, collecting fees on a pair without USDT will result in lost profits as the trading path will be incorrect.
RECOMMENDATION	Ensure that the trading path to USDT is correct.
MITIGATED/COMMENT	The market factory is changed so that all pairs must have USDC as one of the tokens.

No Way To Vote Against Asset End Of Life

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0017
LOCATION	VoteMachine.sol -> 338-341

```
1         if (getEndOfLifeVotes[_symbol].numberOfVotingShares != 0) {
2              endOfLiveValue =
  getEndOfLifeVotes[_symbol].totalVoteValue.div(getEndOfLifeVotes[_symbo
  l].numberOfVotingShares);
3         }
4         else {endOfLiveValue =  0;}
```

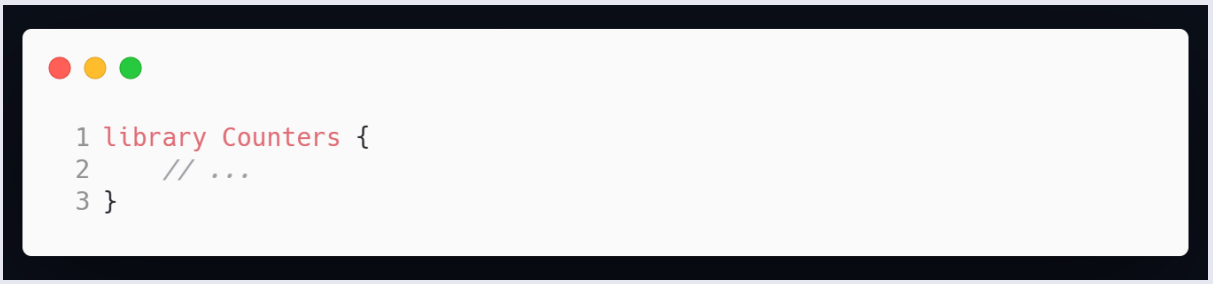
DESCRIPTION	The end of life vote has no mechanism to vote against the asset's life-ending. Anyone can initiate a vote to end the life of an asset. As a result, the project can be attacked by preventing any asset from lasting longer than 2 days.
RECOMMENDATION	Enable users to vote for whether to end the asset's life as well as its price.
MITIGATED/COMMENT	Project team comment: “The initiateEndOfLifeVote function should only be callable when the expiryTime has been reached.”

Local Copies Of OpenZeppelin Contracts

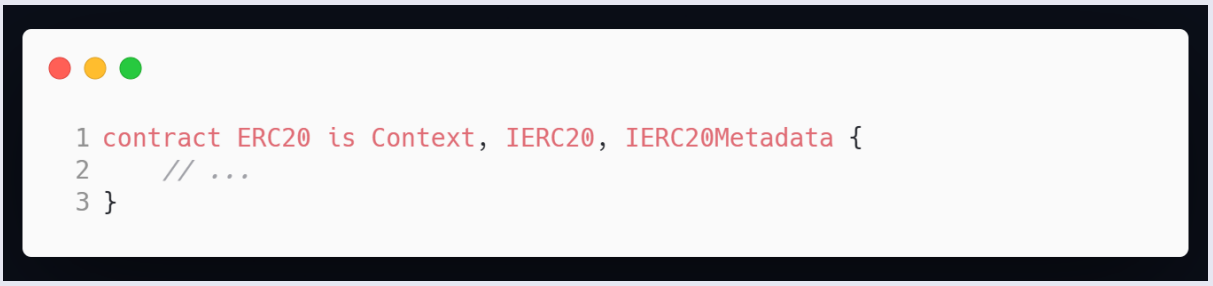
SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0018
LOCATION	Arrays.sol -> 10 Context.sol -> 15 Counters.sol -> 13 ERC20.sol -> 33 ERC20Snapshot.sol -> 33 IERC20.sol -> 8 IERC20Metadata.sol -> 12 Math.sol -> 8 Ownable.sol -> 18 SafeMath.sol -> 15

```
1 library Arrays {
2     // ...
3 }
```

```
1 abstract contract Context {
2     // ...
3 }
```



```
library Counters {
    // ...
}
```

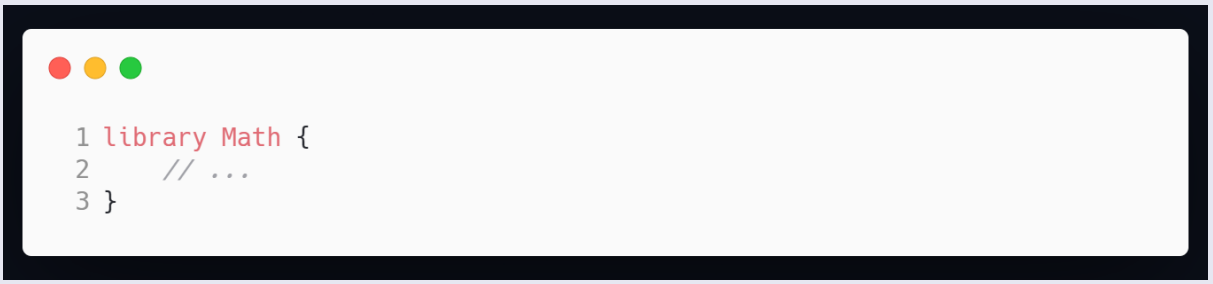


```
contract ERC20 is Context, IERC20, IERC20Metadata {
    // ...
}
```

```
abstract contract ERC20Snapshot is ERC20 {
    // ...
}
```

```
interface IERC20 {
    // ...
}
```

```
interface IERC20Metadata is IERC20 {
    // ...
}
```



```
library Math {
    // ...
}
```

```
abstract contract Ownable is Context {
    // ...
}
```

```
library SafeMath {
    // ...
}
```

DESCRIPTION	The contract includes local copies of OpenZeppelin contracts. The logic within the contracts is mostly identical to OpenZeppelin 4.1. There are some notable differences between the standard OpenZeppelin contracts and the local copies: - ERC20 changes its *_balances*, *_allowances*, *_totalSupply*, and other values from private to internal. - ERC20Snapshot changes *_currentSnapshotId* id from private to internal - Math has two variants of sqrt added.
RECOMMENDATION	Import OpenZeppelin instead of using local copies.
MITIGATED/COMMENT	Project team comment: “Imported Math, Arrays, Context, Ownable, Counters, SafeMath, IERC20 Added a new separate Math2 contract, which contains the square function ERC20 and ERC20Snapshot have been left with local copies as we wanted to keep the adapted mint function, which is a direct copy from uniswap - and thus we needed to change the visibility of some parameters from private to internal.”

Direct Modification Of ERC20 Values

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0019
LOCATION	MarketERC20.sol -> 53-59

```
1     function _mint(address account, uint256 amount) internal override
  {
2         _beforeTokenTransfer(address(0), account, amount);
3
4         _totalSupply += amount;
5         _balances[account] += amount;
6         emit Transfer(address(0), account, amount);
7     }
```

DESCRIPTION	Contract directly modifies internal values of the modified ERC20 contract.
RECOMMENDATION	Derived implementation should call parent contract's *_mint* instead of modifying values meant to be private.
MITIGATED/COMMENT	Project team comment: “ERC20 and ERC20Snapshot have been left with local copies as we wanted to keep the adapted mint function, which is a direct copy from uniswap - and thus we needed to change the visibility of some parameters from private to internal.”

Local Copies Of Uniswap Libraries

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0020
LOCATION	libraries/MarketLibrary.sol -> 8 libraries/Math.sol -> 7 libraries/TransferHelper.sol -> 7

```
1 library MarketLibrary {
2     // ...
3 }
```

```
1 library Math {
2     // ...
3 }
```

```
1 library TransferHelper {
2     // ...
3 }
```

DESCRIPTION	The contract includes local copies of uniswap-lib and uniswap-v2-periphery contracts. The logic is identical to uniswap-lib 2.1.0 and uniswap-v2-periphery 1.0.0.
RECOMMENDATION	Import uniswap-lib and uniswap-v2-periphery instead of using local copies.
MITIGATED/COMMENT	Project team comment: “We have done some significant changes to the uniswap smart contracts, we thus prefer to keep them as they are”

Incorrect Error Message

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0021
LOCATION	Counters.sol -> 31-37

```
1     function decrement(Counter storage counter) internal {
2         uint256 value = counter._value;
3         require(value > 0, "Counter: decrement overflow");
4         unchecked {
5             counter._value = value - 1;
6         }
7     }
```

DESCRIPTION	If the value is zero and is decreased by 1, it's called underflow.
RECOMMENDATION	Change the message from overflow to underflow
MITIGATED/COMMENT	Counters.sol is now imported from Openzeppelin.

SafeMath Unnecessary In Solidity 0.8

SEVERITY	Informational
RESOLVED	**NO**
FINDING ID	#0022
LOCATION	assetFactory.sol -> 19 DAO.sol -> 13 GovernanceToken.sol -> 8 MarketERC20.sol -> 17 MarketPair.sol -> 19 MarketLibrary.sol -> 9 MarketRouter.sol -> 23 RewardsMachine.sol -> 15 VoteMachine.sol -> 12

```
1     using SafeMath for uint256;
2
```

```
1     using SafeMath for uint;
2
```

DESCRIPTION	Overflow and underflow are checked implicitly in solidity 0.8. Using SafeMath is both redundant and costs extra gas.
RECOMMENDATION	Replace SafeMath with built-in operators.
MITIGATED/COMMENT	N/A

Duplicate Contracts

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0023
LOCATION	interfaces/IERC20I.sol -> 3 openzeppelin/IERC20.sol -> 8

```
1 interface IERC20I {
2     // ...
3 }
```

```
1 interface IERC20 {
2     // ...
3 }
```

DESCRIPTION	These contracts are identical other than their name and code comments.
RECOMMENDATION	Since these are both OpenZeppelin contracts, they should be replaced with an import.
MITIGATED/COMMENT	interfaces/IERC20I.sol remains openzeppelin/IERC20.sol removed. Project team comment: “Acknowledged, but given that this is no security issue we keep them as a change now would come with the risk of adding mistakes ;-)”

Duplicate Contract Names

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0024
LOCATION	libraries/Math.sol -> 7 openzeppelin/Math.sol -> 8

```
1 library Math {
2     // ...
3 }
```

```
1 library Math {
2     // ...
3 }
```

DESCRIPTION	Duplicate contract names may cause issues with compilation.
RECOMMENDATION	Rename the duplicate contracts.
MITIGATED/COMMENT	N/A

Unused Contract Values

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0025
LOCATION	MarketERC20.sol -> 27-29 MarketERC20.sol -> 25 MarketPair.sol -> 27-28

```
1     bytes32 public DOMAIN_SEPARATOR;
2     bytes32 public constant PERMIT_TYPEHASH =
  0x6e71edae12b1b97f4d1f60370fef10105fa2faae0126114a169c64845d6126c9;
3     mapping(address => uint256) public nonces;
```

```
1     address[] public holders;
2
```

```
1     uint32  private blockTimestampLast;
2     address INTAddress;
3     address tokenFactoryAddress;
```

DESCRIPTION	Contract values are either never set or used anywhere. Some additional notes: - MarketERC20.holders is set by MarketERC20.addHolder and MarketERC20.removeHolder, which are never called. - MarketPair.blockTimestampLast is returned by MarketPair.getReserves but never set.
RECOMMENDATION	Use these values or remove them and their associated getters/setters.
MITIGATED/COMMENT	Unused variables were removed.

Redundant Functions (Gas Optimization)

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0026
LOCATION	MarketRouter.sol -> 104-114 MarketFactory.sol -> 63-74

```
1     function getMarketPair(
2         address tokenA,
3         address tokenB
4         )
5         public
6         view
7         returns (address)
8         {
9         address pair = MarketFactory(factory).getMarketPair(tokenA,
  tokenB);
10        return pair;
11    }
```

```
1     function getMarketPair(
2         address tokenA,
3         address tokenB
4         )
5         external
6         override
7         view
8         returns (address pair)
9         {
10        pair = getPair[tokenA][tokenB];
11        return pair;
12    }
```

DESCRIPTION	The function *MarketRouter.getMarketPair()* redirects to *MarketFactory.getMarketPair()* which in turn redirects to *MarketFactory.getPair()*.
RECOMMENDATION	Replace instances of these functions with *MarketFactory.getPair()*.
MITIGATED/COMMENT	Redundant functions were removed.

Duplicate Functionality

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0027
LOCATION	MarketRouter.sol MarketLibrary.sol

DESCRIPTION	The following functions have duplicate implementations between MarketRouter and MarketLibrary: The following functions redirect to MarketLibrary functions: - MarketRouter.quote - MarketRouter.getAmountIn - MarketRouter.getAmountsIn
RECOMMENDATION	Remove duplicate implementations and redirecting functions and use MarketLibrary.
MITIGATED/COMMENT	Duplicate functions were removed from the library.

MarketRouter	MarketLibrary
`sortPairTokens`	`sortTokens`
`getPairReserves`	`getReserves`
`getAmountsOut`	`getAmountsOut`
`getAmountOut`	`getAmountOut`

Interface Does Not Match Contract

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0028
LOCATION	MarketPair.sol IMarketPair.sol

DESCRIPTION	The following functions are listed in the interface but not implemented in the contract: - IMarketPair.permit - IMarketPair.price0CumulativeLast - IMarketPair.price1CumulativeLast - IMarketPair.skim - IMarketPair.sync The following function does not match the implementation in the contract: - IMarketPair.initialize
RECOMMENDATION	Update the contract or the interface to match each other.
MITIGATED/COMMENT	The interface was updated.

Interface Is Empty

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0029
LOCATION	IMarketRouter02.sol -> 6-8

```
1 interface IMarketRouter02 is IMarketRouter01 {
2
3 }
```

DESCRIPTION	This interface has no content.
RECOMMENDATION	Remove this interface.
MITIGATED/COMMENT	References to the interface were removed.

Unused Constructor Parameters

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0030
LOCATION	AssetToken.sol -> 11-19

```
1     constructor (
2         string memory _name,
3         string memory _symbol,
4         string memory _description,
5         uint256 _upperLimit
6         )
7
8         ERC20(_name,_symbol)
9         {}
```

DESCRIPTION	The constructor parameters *_description* and *_upperLimit* are not used.
RECOMMENDATION	Confirm that this is the expected behavior. Remove the parameters if it isn't.
MITIGATED/COMMENT	Parameters were removed from constructor.

Address Indicates USDC Instead Of USDT

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0031
LOCATION	assetFactory.sol -> 24

```
1     address private USDTaddress =
  0x2791Bca1f2de4661ED88A30C99A7a9449Aa84174;
2
```

DESCRIPTION	The address assigned to *USDTaddress* refers to USDC as opposed to USDT.
RECOMMENDATION	Rename the value or correct the address.
MITIGATED/COMMENT	Variable was renamed to USDCaddress.

Struct Value Not Assigned

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0032
LOCATION	assetFactory.sol -> 122-142

```
1     function createAssets (
2         string calldata _name,
3         string calldata _symbol,
4         string calldata _description,
5         uint256 _upperLimit
6         )
7         external
8         onlyOwner
9         {
10        //...
11    }
```

DESCRIPTION	The following value *issuaaLibrary.Asset.symbol* is never set nor used.
RECOMMENDATION	Remove the value.
MITIGATED/COMMENT	Value was removed.

Function Does Not Burn Tokens

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0033
LOCATION	assetFactory.sol -> 296-308

```
1     function burnGovernanceToken (
2         uint256 _amount
3         )
4         external
5         {
6
7         uint256 USDTAmount =
  (feePool.mul(_amount)).div(RewardsMachine(rewardsMachineAddress).maxIN
  TSupply());
8         feePool = feePool.sub(USDTAmount);
9         _transferFrom(INTAddress,msg.sender, address(this), _amount);
10
   RewardsMachine(rewardsMachineAddress).reduceCurrentINTSupply(_amount)
  ;
11        //governanceToken.burn(address(this), _amount);
12        IERC20(USDTaddress).transfer(msg.sender,USDTAmount);
13    }
```

DESCRIPTION	The noted function is described as a burn function however it only transfers governance tokens to itself, without burning them.
RECOMMENDATION	Change the function name or behavior to match.
MITIGATED/COMMENT	*assetFactory.burnGovernanceToken* was modified to call the burn function.

Rewards Are Lost If Not Collected Every Period

SEVERITY	Informational
RESOLVED	**SEE COMMENT**
FINDING ID	#0034
LOCATION	RewardMachine.sol -> 185-234 VoteMachine.sol -> 474-479

```
1     function claimRewards()
2         external
3         returns (uint256)
4         {
5             require (lastRewardsRound[msg.sender]
  <rewardsRound,'CLAIMED_ALREADY');
6             lastRewardsRound[msg.sender] = rewardsRound;
7             // ...
8         }
```

```
1     function resetRewardPoints ()
2         external
3         {
4             require (msg.sender ==
  rewardsMachineAddress,'NOT_ALLOWED');
5             currentRewardsRound = currentRewardsRound +1;
6     }
```

DESCRIPTION	Each user may only collect rewards for the latest reward period. These rewards are not cumulative and will therefore be lost.
RECOMMENDATION	Confirm that this behavior is expected and ensure that it is documented for users. Alternatively, change the lastRewardsRound to increment by one to allow users to collect for multiple periods/snapshots.
MITIGATED/COMMENT	Project team comment: “This is indeed a feature not a bug as we want people to come back to the platform in regular intervals”

Rewards Labeled As Weekly But Calculated Daily

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0035
LOCATION	RewardMachine.sol -> 162-179

```
1    function createRewards()
2        external
3        {
4        require(nextRewardsPayment<block.timestamp,'TIME_NOT_UP');
5        // ...
6        nextRewardsPayment = block.timestamp.add(1 days);
7        // ...
8    }
```

DESCRIPTION	Rewards are named as calculated weekly but can be calculated daily.
RECOMMENDATION	Change the time or the variable name to be consistent.
MITIGATED/COMMENT	Rewards are now calculated every 7 days.

Maximum Vesting Schedules Vulnerability

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0041
LOCATION	Revision-2: GovernanceToken.sol -> 104-121

```
1   function transferAndVest(
2     address _address,
3     uint256 _amount,
4     uint256 _time
5     )
6     external
7     {
8       require (_time < 731 days,"VESTING_PERIOD_TOO_LONG");
9       require
  (vestingSchedules[_address].length<maxVestingEntries,"TOO_M
  ANY_VESTING_ENTRIES");
10      _burn(msg.sender, _amount);
11      if (stakes[_address] == 0) {
12        isStakeholder[_address] = true;
13        numberOfStakeholders = numberOfStakeholders + 1;
14      }
15      stakes[_address] = stakes[_address].add(_amount);
16
   vestingSchedules[_address].push([block.timestamp.add(_time
  ),_amount]);
17
18    }
```

DESCRIPTION	Users cannot receive vested rewards if they reach the maximum number of vesting schedules. Because `transferAndVest` has no checks on the caller, a user can have their vesting schedules increased to the maximum by any malicious actor. A user who receives many rewards can also potentially lock themselves out during normal contract operations.

RECOMMENDATION	Use a different vesting mechanism to limit the looping.
MITIGATED/COMMENT	Vesting via transfers is now limited to the first 10 vesting schedules for any given account. There is also now a minimum amount to be vested using such a method. Project team comment: “This function is supposed to be used for claiming functions from a (yet to be written) contract that allows people that have earned IPT token before the mainnet launch. Later on, this function will not be necessary anymore.”

Cannot Claim Rewards When Losing A Vote

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0042
LOCATION	Revision-2: RewardsMachine.sol -> 206-260

```
1     function claimRewards()
2         external
3         returns (uint256)
4         {
5             require (lastRewardsRound[msg.sender]
  <rewardsRound-1,'CLAIMED_ALREADY');
6
   VoteMachine(voteMachineAddress).checkVotesIfClosed(rewards
  Round - 1,msg.sender);
7
   require(VoteMachine(voteMachineAddress).checkFreezeVotes(r
  ewardsRound - 1,msg.sender),'VOTE_NOT_CONSENSUS');
8
9
   require(VoteMachine(voteMachineAddress).checkExpiryVotes(r
  ewardsRound - 1,msg.sender),'VOTE_NOT_CONSENSUS');
10            // ...
11        }
```

DESCRIPTION	No rewards can be claimed for a reward round if the user voted against the result of any vote which closed this round. Both VoteMachine.checkFreezeVotes and VoteMachine.checkExpiryVotes return false if the user voted against any result.
RECOMMENDATION	Change the check to reduce the reward based on the voting result instead of blocking all rewards.
MITIGATED/COMMENT	Project team comment: “We have discussed the issue

before we made the changes, and the current behavior is indeed the desired outcome. There are now only two kinds of voting processes to be checked if your vote was consensus or not - freezeVotes and ExpiryVotes. These votes function as oracles, the outcome should thus be objective rather than subjective. Taking away all rewards from users that vote against the consensus vote is thus one of the features which help to avoid people voting and trying to manipulate the price.”

Cannot Burn Tokens Marked Both Frozen And Expired

SEVERITY	Low Risk
RESOLVED	**YES**
FINDING ID	#0043
LOCATION	Revision-3: assetFactory.sol -> 315-334 assetFactory.sol -> 342-361

```
1     function burnAssets (
2         string calldata _symbol,
3         uint256 _amount
4         )
5         external
6         {
7         require(getAsset[_symbol].expired ==
  false,'EXPIRED');
8         // ...
9     }
```

```
    function burnExpiredAssets (
        string calldata _symbol,
        uint256 _amount1,
        uint256 _amount2
        )
        external
        {
        require(getAsset[_symbol].expired == true,'NOT_EXPIRED');
        require(getAsset[_symbol].frozen == false,'FROZEN');
        // ...
    }
```

DESCRIPTION	Expired assets can only be redeemed if not frozen. In normal operation, assets cannot be frozen after they are expired and frozen assets cannot be expired. However, near the expiry time of a token, it will be possible to initiate a vote to freeze the token. Then, once the expiry time is reached, initiate a vote to expire the token. This can result in tokens that can no longer be redeemed.
RECOMMENDATION	Handle this edge case in the close vote functions or provide an alternative way to redeem said tokens.
MITIGATED/COMMENT	RewardMachine was changed so that end-of-life votes can only be started if no freeze vote is open.

Static Analysis

Division Before Multiplication

SEVERITY	Medium Risk
RESOLVED	**YES**
FINDING ID	#0036
LOCATION	assetFactory.sol -> 234-235 assetFactory.sol -> 251-253 assetFactory.sol -> 281-286 VoteMachine.sol -> 339 VoteMachine.sol -> 345-349 RewardsMachine.sol -> 166-168 RewardsMachine.sol -> 215-218 RewardsMachine.sol -> 271-274

```
1         uint256 USDDecimals = ERC20(USDTaddress).decimals();
2         uint256 tokenAmount = _amount.mul(10**(18-
  USDDecimals)).div(getAsset[_symbol].upperLimit/1000);
```

```
1         uint256 USDDecimals = ERC20(USDTaddress).decimals();
2         uint256 upperLimit = getAsset[_symbol].upperLimit.div(1000);
3         uint256 amountOut = _amount.mul(upperLimit).div(10**(18-
  USDDecimals));
```

```
1         uint256 USDDecimals = ERC20(USDTaddress).decimals();
2         uint256 endOfLifeValue =
  (getAsset[_symbol].endOfLifeValue).div(1000);
3         uint256 upperLimit = (getAsset[_symbol].upperLimit).div(1000);
4         uint256 valueShort = upperLimit.sub(endOfLifeValue);
5         uint256 amountOut1 = _amount1.mul(endOfLifeValue).div(10**(18-
  USDDecimals));
6         uint256 amountOut2 = _amount2.mul(valueShort).div(10**(18-
  USDDecimals));
```

```
    endOfLiveValue = getEndOfLifeVotes[_symbol].totalVoteValue.div(getEndOfLifeVotes[_symbol].numberOfVotingShares);

```

```
    if (
        getEndOfLifeVotes[_symbol].individualVotes[i].voteValue > endOfLiveValue.mul(99).div(100)
        &&
        getEndOfLifeVotes[_symbol].individualVotes[i].voteValue < endOfLiveValue.mul(101).div(100)
    )
```

```
    uint256 weeklyRewards = maxINTSupply.sub(currentINTSupply).div(20);
    votingRewardTokenNumber = weeklyRewards.mul(20).div(100);
    LPRewardTokenNumber = weeklyRewards.mul(80).div(100);
```

```
    uint256 poolRewards = LPRewardTokenNumber.div(numberOfPools);
    if (LPTokenTotalSupply >0){
        rewards = poolRewards.mul(LPTokenBalance).div(LPTokenTotalSupply);
    }
```

```
    uint256 poolRewards = LPRewardTokenNumber.div(numberOfPools);
    if (LPTokenTotalSupply >0){
      rewards = poolRewards.mul(LPTokenBalance).div(LPTokenTotalSupply);
    }
```

DESCRIPTION	The calculation for the amounts noted above uses mixed orders of multiplication and division. This may result in rounding errors.

	The rounding errors can potentially cause transactions to revert if attempting to transfer the remaining tokens in the contract and the amounts round up. The rounding errors can also create a discrepancy between amounts minted and burned.
RECOMMENDATION	Change the calculations to first multiply, then divide. Ensure that the USDT balance always exceeds the amount to be transferred.
MITIGATED/COMMENT	All calculations were modified to multiply before division.

No Events Emitted For Changes To Protocol Values

SEVERITY	Informational
RESOLVED	**PARTIAL**
FINDING ID	#0037
LOCATION	• assetFactory.sol -> 67-74: function setVoteMachineAddress(address _address) external onlyOwner • assetFactory.sol -> 80-87: function setRewardsMachineAddress(address _address) external onlyOwner • assetFactory.sol -> 93-100: function setMarketFactoryAddress(address _address) external onlyOwner • assetFactory.sol -> 106-113: function setMarketRouterAddress(address _address) external onlyOwner • assetFactory.sol -> 314-321: function freezeAsset(string calldata _symbol) external • assetFactory.sol -> 328-337: function setEndOfLifeValue(string calldata _symbol, uint256 _value) external • DAO.sol -> 84-91: function setVoteMachineAddress(address _address) external onlyOwner • DAO.sol -> 98-105: function setRewardsMachineAddress(address _address) external onlyOwner • DAO.sol -> 111-118: function setAssetFactorAddress(address _address) external onlyOwner • DAO.sol -> 127-151: function initiateGrantFundingVote(address _receiver,uint256 _amount,string calldata _description) external • DAO.sol -> 210-244: function closeGrantFundingVote(address _receiver) external • DAO.sol -> 280-306: function initiateNewAssetVote(string calldata _symbol,string calldata _name,uint256 _upperLimit,string calldata _description) external • DAO.sol -> 364-399: function closeNewAssetVote(string calldata _symbol) external • GovernanceToken.sol -> 36-43: function setVoteMachineAddress(address _address) external onlyOwner • GovernanceToken.sol -> 49-56: function setDAOAddress(address _address) external onlyOwner • MarketERC20.sol -> 83-90: function addHolder(address _address) internal

	• MarketERC20.sol -> 96-105: function removeHolder(address _address) internal • MarketFactory.sol -> 36-43: function setRewardsMachineAddress (address _address) external onlyOwner • MarketFactory.sol -> 114-122: function setFeeTo(address _feeTo) external override • MarketFactory.sol -> 128-136: function setFeeToSetter(address _feeToSetter) external override • RewardMachine.sol -> 57-64: function setVoteMachineAddress(address _address) external onlyOwner • RewardMachine.sol -> 70-77: function setMarketFactoryAddress(address _address) external onlyOwner • RewardMachine.sol -> 83-90: function setAssetFactoryAddress(address _address) external onlyOwner • RewardMachine.sol -> 100-107: function reduceCurrentINTSupply(uint256 _amount) external • RewardMachine.sol -> 124-139: function addPools(string calldata _symbol) external onlyOwner • RewardMachine.sol -> 145-156: function addIPTBonusPool(address _poolAddress) external • VoteMachine.sol -> 108-115: function setAssetFactoryAddress(address _address) external onlyOwner • VoteMachine.sol -> 122-129: function setRewardsMachineAddress(address _address) external onlyOwner • VoteMachine.sol -> 135-142: function setDAOAddress(address _address) external onlyOwner • VoteMachine.sol -> 148-151: function initiateFreezeVote(string calldata _symbol) external • VoteMachine.sol -> 236-268: function closeFreezeVote (string calldata _symbol) external

DESCRIPTION	Functions that change important variables should include emit logs such that users can more easily monitor the change.
RECOMMENDATION	Add emit logs to these functions. Ensure that these values are secured via a timelock.
MITIGATED/COMMENT	Project team comment: “For the function that are setting the addresses there are in our view no events necessary.

	The smart contract is owned by the DAO smart contract, which has no function implemented that can call these functions after the initial deployment." Obelisk comment: "Ownership of contracts will be checked on-chain. Some variables can still be set by the contract owner."

Missing Zero Checks

SEVERITY	Informational
RESOLVED	**SEE COMMENT**
FINDING ID	#0038
LOCATION	• assetFactory.sol -> 34-42: constructor(address governanceTokenAddress, address _tokenFactoryAddress) Ownable() • assetFactory.sol -> 67-74: function setVoteMachineAddress (address _address) external onlyOwner • assetFactory.sol -> 80-87: function setRewardsMachineAddress (address _address) external onlyOwner • assetFactory.sol -> 93-100: function setMarketFactoryAddress (address _address) external onlyOwner • assetFactory.sol -> 106-113: function setMarketRouterAddress (address _address) external onlyOwner • DAO.sol -> 74-78: constructor(GovernanceToken _governanceToken, VoteMachine _voteMachine, AssetFactory _assetFactory) • DAO.sol -> 84-91: function setVoteMachineAddress (address _address) external onlyOwner • DAO.sol -> 98-105: function setRewardsMachineAddress (address _address) external onlyOwner • DAO.sol -> 111-118: function setAssetFactorAddress (address _address) external onlyOwner • GovernanceToken.sol -> 36-43: function setVoteMachineAddress(address _address) external onlyOwner • GovernanceToken.sol -> 49-56: function setDAOAddress(address _address) external onlyOwner • RewardsMachine.sol -> 46-51: constructor(GovernanceToken _governanceToken) • RewardsMachine.sol -> 57-64: function setVoteMachineAddress (address _address) public onlyOwner • RewardsMachine.sol -> 70-77: function setMarketFactoryAddress (address _address) public onlyOwner • RewardsMachine.sol -> 82-90: function setAssetFactoryAddress (address _address) public onlyOwner • RewardsMachine.sol -> 145-156: function addIPTBonusPool(address _poolAddress) external • VoteMachine.sol -> 99-102: constructor(GovernanceToken _governanceToken, AssetFactory _assetFactory)

	• VoteMachine.sol -> 108-115: function setAssetFactoryAddress (address _address) external onlyOwner • VoteMachine.sol -> 122-129: function setRewardsMachineAddress (address _address) external onlyOwner • VoteMachine.sol -> 132-142: function setDAOAddress (address _address) external onlyOwner

DESCRIPTION	Functions don't check for a zero address before assigning variables.
RECOMMENDATION	Add a check for zero address if deemed necessary.
MITIGATED/COMMENT	Project team comment: “Given these are all only set once, a specific check is in our view not necessary”

Unused Variables

SEVERITY	Informational
RESOLVED	**YES**
FINDING ID	#0039
LOCATION	• DAO.sol -> 15: address voteMachineAddress; • DAO.sol -> 16: address rewardsMachineAddress; • DAO.sol -> 17: address assetFactoryAddress; • GovernanceToken.sol -> 19: mapping(address => uint256) public totalEscrowedAccountBalance; • GovernanceToken.sol -> 21: mapping(address => uint256) public totalVestedAccountBalance; • GovernanceToken.sol -> 24: uint256 public totalEscrowedBalance; • RewardMachine.sol -> 27: address public master; • RewardMachine.sol -> 32: uint256 public assetNumber; • RewardMachine.sol -> 34: string[] public assets; • RewardMachine.sol -> 37: uint256 public stakingRewardPoints; • RewardMachine.sol -> 43: mapping(address => uint256) public stakingRewardsPoints; • RewardMachine.sol -> 44: address[] public stakers; • RewardMachine.sol -> 97: mapping(address => grantFundingVotes) public getGrantVotes;

DESCRIPTION	These variables are not used anywhere in the contract.
RECOMMENDATION	Remove the variables.
MITIGATED/COMMENT	Unused variables were removed.

Variable Can Be Declared Constant Or Immutable (Gas Optimization)

SEVERITY	Informational
RESOLVED	**PARTIAL**
FINDING ID	#0040
LOCATION	• assetFactory.sol -> 24: address private USDTaddress = 0x2791Bca1f2de4661ED88A30C99A7a9449Aa84174; • DAO.sol -> 24: uint256 DAOVolume • RewardsMachine.sol -> 18: uint256 public maxINTSupply • RewardsMachine.sol -> 28: address private USDTaddress • RewardsMachine.sol -> 23: GovernanceToken public governanceToken • RewardsMachine.sol -> 33: uint256 public vestingPeriod • VoteMachine.sol -> 17: GovernanceToken public governanceToken • VoteMachine.sol -> 18: uint256 DAOVolume

DESCRIPTION	*USDTaddress* is an address that is being initialized. Should be declared constant to avoid unintended behavior.
RECOMMENDATION	Declare variables as constant or immutable.
MITIGATED/COMMENT	The following variables can still be made constant or immutable: - RewardsMachine.sol -> 23: GovernanceToken public governanceToken - VoteMachine.sol -> 17: GovernanceToken public governanceToken - VoteMachine.sol -> 18: uint256 DAOVolume

On-Chain Analysis

Unverified Contracts

SEVERITY	Informational
RESOLVED	**Yes**
FINDING ID	#0044
LOCATION	AssetToken.sol 0xBb66834dAE8D8B7cd9F127e6d76280827A41ccb7 0xdb83dA2Ff76D9530EdBA66683FD6a7b4c425Bdd6 0xe6395A08FF09D5F04150913E19a9c7F530841c70 0xf30aBEd397082bbdA59eF8264d3e4923e5FE09e1 0x401b4C44A133A940e483a8F54Bba853419466cb0 0x4E15F27f49d8c9Afd08A9a793d13524Aa76c37b5 0xe0136c1750Fc00d138885319885453A98b5Cd07c 0xb6E2348c4C22c129f52541C715188BfB944d1Cf1 0x168178fC734E5aC7cE4Ab3d6720D5CaD85d9f43D 0x27c156Fd4f4139F391fd3959d07aC66E1c4e876b 0xA45419fde5483892504b3f8C5a9AfFF150417126 0xFA8a2807cAa3446591f8CB7c3Ee83C9BCF7D3F8a 0x1d2e9C419B3bd4e32B1f943EF2a41bf43a52520f 0xb028EA433571DEebb90E4503D7A6eE35be4a5614 0x81aDb920E3D14c66264fA16E6726E852fDcA98f7 0xc2F6574297cC9b36ca66b2968A2aDa987bEdf40E MarketPair.sol 0x80a6A17662c7737E3Bf6095759014aa4b7c1CDA3 0xd49Bd7F84fff5B7C767577b48093f72793D3b7b4 0xd4EFec6C8db4FBe68b13daB8113E72B12CbB23de 0x0F4D127c6F192601fb25c180F7b738D239e895f0 0x5fB203F1B9F623A0f4f194bC0263F33B438E150A 0x71b20688dF4713f33A37E91139e9aB7A9eD23f91 0x386E0E441a33745914BE3852f114b729B184f730 0xfd69841BB87f8909dAa7797c88E929E65EeAf7F3 0x562849B385F82a705B6A9A0477D9D6d690D8D7f8 0x6D658d7Ed51664489c0CD088448a929a8fD54b78 0xc4c7a2dfD0E6C74431691275c95B2e57C806F76C 0xD624AA3c2bf738ddb9c337C48fa104Ae3E13fD4D 0x645f7019a27bd014E783AfFB1Bc19f1aF6c37E2b 0xDB046fa2A3CFedD0614e1935eB4A032329BD1b30 0x9C7e44b0417c0aCd36F3C1052dA20720F2351001 0x8F9D8dD5257aaB6f2de6755d5219796cedcaFb5E 0x5a77378Ee76cb02D28CE737B6bde53e12EFE692e

DESCRIPTION	Contracts deployed by existing contracts are not verified by default.
RECOMMENDATION	Verify these contracts separately.
MITIGATED/COMMENT	Contracts were verified

Appendix A - Reviewed Documents

Document	Address
interfaces/IERC20I.sol	N/A
interfaces/IMarketPair.sol	N/A
interfaces/IMarketFactory.sol	N/A
interfaces/IMarketRouter01.sol	N/A
interfaces/IMarketRouter02.sol	N/A
libraries/Math.sol	N/A
libraries/MarketLibrary.sol	N/A
libraries/TransferHelper.sol	N/A
openzeppelin/Array.sol	N/A
openzeppelin/Context.sol	N/A
openzeppelin/Counters.sol	N/A
openzeppelin/IERC20.sol	N/A
openzeppelin/IERC20Metadata.sol	N/A
openzeppelin/Math.sol	N/A
openzeppelin/Ownable.sol	N/A
openzeppelin/SafeMath.sol	N/A
openzeppelin/ERC20.sol	N/A
openzeppelin/ERC20Snapshot.sol	N/A
assetFactory.sol	0x0d9bF3A539C515378b70A7bdDb1aa712521Abc9F

AssetToken.sol	DJIA_2209 0xBb66834dAE8D8B7cd9F127e6d76280827A41ccb7 0xdb83dA2Ff76D9530EdBA66683FD6a7b4c425Bdd6 NDX_2209 0xe6395A08FF09D5F04150913E19a9c7F530841c70 0xf30aBEd397082bbdA59eF8264d3e4923e5FE09e1 S500_2209 0x401b4C44A133A940e483a8F54Bba853419466cb0 0x4E15F27f49d8c9Afd08A9a793d13524Aa76c37b5 WTI_2209 0xe0136c1750Fc00d138885319885453A98b5Cd07c 0xb6E2348c4C22c129f52541C715188BfB944d1Cf1 XAU_2209 0x168178fC734E5aC7cE4Ab3d6720D5CaD85d9f43D 0x27c156Fd4f4139F391fd3959d07aC66E1c4e876b XAG_2209 0xA45419fde5483892504b3f8C5a9AfFF150417126 0xFA8a2807cAa3446591f8CB7c3Ee83C9BCF7D3F8a BTC_2209 0x1d2e9C419B3bd4e32B1f943EF2a41bf43a52520f 0xb028EA433571DEebb90E4503D7A6eE35be4a5614 ETH_2209 0x81aDb920E3D14c66264fA16E6726E852fDcA98f7 0xc2F6574297cC9b36ca66b2968A2aDa987bEdf40E
DAO.sol	0x2dE3c01f5a1f2373d88fb72596ebA5155Da9Aafe
GovernanceToken.sol	0x95f91c82EdBa4D2B985f9435E3791191a4289F45
issuaaLibrary.sol	N/A
MarketFactory.sol	0xe41BD99D3288fEb1CAC32276d172eae3BB660790
MarketRouter.sol	0x099aA2fb3d65e3dAA9bB2d5f70d0ae718C7e6d73
MarketERC20.sol	N/A
MarketPair.sol	0x80a6A17662c7737E3Bf6095759014aa4b7c1CDA3 0xd49Bd7F84fff5B7C767577b48093f72793D3b7b4 0xd4EFec6C8db4FBe68b13daB8113E72B12CbB23de 0x0F4D127c6F192601fb25c180F7b738D239e895f0 0x5fB203F1B9F623A0f4f194bC0263F33B438E150A

	0x71b20688dF4713f33A37E91139e9aB7A9eD23f91 0x386E0E441a33745914BE3852f114b729B184f730 0xfd69841BB87f8909dAa7797c88E929E65EeAf7F3 0x562849B385F82a705B6A9A0477D9D6d690D8D7f8 0x6D658d7Ed51664489c0CD088448a929a8fD54b78 0xc4c7a2dfD0E6C74431691275c95B2e57C806F76C 0xD624AA3c2bf738ddb9c337C48fa104Ae3E13fD4D 0x645f7019a27bd014E783AfFB1Bc19f1aF6c37E2b 0xDB046fa2A3CFedD0614e1935eB4A032329BD1b30 0x9C7e44b0417c0aCd36F3C1052dA20720F2351001 0x8F9D8dD5257aaB6f2de6755d5219796cedcaFb5E 0x5a77378Ee76cb02D28CE737B6bde53e12EFE692e
Migrations.sol	N/A
RewardsMachine.sol	0xd4C1bc44423AA8B84d2271F5aD177ae32457f500
TokenFactory.sol	0xCBC3b620c91aef80Be7a6ff164149E9DA0F62385
VoteMachine.sol	0x34c3896267aC5B7D4FC3DF0BeFd8087244f78a71

Appendix B - Risk Ratings

Risk	Description
High Risk	A fatal vulnerability that can cause immediate loss of Tokens / Funds
Medium Risk	A vulnerability that can cause some loss of Tokens / Funds
Low Risk	A vulnerability that can be mitigated
Informational	No vulnerability

Appendix C - Testing Standard

An ordinary audit is conducted using these steps.

1. Gather all information
2. Conduct a first visual inspection of documents and contracts
3. Go through all functions of the contract manually (2 independent auditors)
    a. Discuss findings
4. Use specialized tools to find security flaws
    a. Discuss findings
5. Follow up with project lead of findings
6. If there are flaws, and they are corrected, restart from step 2
7. Write and publish a report

During our audit, a thorough investigation has been conducted employing both automated analysis and manual inspection techniques. Our auditing method lays a particular focus on the following important concepts:

- Ensuring that the code and codebase use best practices, industry standards, and available libraries.
- Testing the contract from different angles ensuring that it works under a multitude of circumstances.
- Analyzing the contracts through databases of common security flaws.

Follow Obelisk Auditing for the Latest Information

 ObeliskOrg

 ObeliskOrg

OBELISK

Part of Tibereum Group